(2)

WORKING MATERIAL

for the lectures of

Zohar Manna

Program Synthesis

International Summer School

on

LOGIC, ALGEBRA AND COMPUTATION

Marktoberdorf, Germany, July 25 - August 6, 1989

This Summer School is organized under the auspices of the Technische Universität München and is sponsored by the NATO Science Committee as part of the 1988 Advanced Study Institutes programme. partial support for the conference was provided by the European Research Office and the National Science Foundation and by various industrial companies.

(3)

ZOHAR MANNA and RICHARD WALDINGER

DRAFT

June 1989

Program Synthesis

Up to now in this book, we have defined a function by introducing a set of axioms. Typically these axioms are computationally suggestive, that is, they have suggested a method for computing the function. But it is not always obvious that the axioms define the function we intend. From these axioms, we have established properties of the function they define which gives us some assurance that that function is indeed the correct one.

For example, we may define the greatest-common-divisor function $gcd(x_1, x_2)$ by the axiom (see Section [I]-0.11)

$$(\forall\ integer\ x_1, x_2)\left[gcd(x_1, x_2) = \begin{cases} if\ x_2 = 0 \\ then\ x_1 \\ else\ gcd(x_2, rem(x_1, x_2)) \end{cases}\right]$$

This axiom is computationally suggestive, but it is by no means obvious that the function it defines is in fact the greatest common divisor.

From this axiom, however, we can prove the *greatest-common-divisor* property

$$(\forall\ \textit{integer}\ x_1,\ x_2)\begin{bmatrix} gcd(x_1,\ x_2) \preceq_{div} x_1 \quad \textit{and} \quad gcd(x_1, x_2) \preceq_{div} x_2 \\ \quad \textit{and} \\ (\forall\ \textit{integer}\ y)\begin{bmatrix} \textit{if}\ \ y \preceq_{div} x_1\ \ \textit{and}\ \ y \preceq_{div} x_2 \\ \textit{then}\ \ y \preceq_{div} gcd(x_1,\ x_2) \end{bmatrix} \end{bmatrix}.$$

⟪ well-founded induction must apply to tuples of different sorts as in §14.7 m-tuples of sort $\overline{\textit{i-obj}}$ ⟫

In short, $gcd(x_1,\ x_2)$ is the "greatest" nonnegative integer that divides both x_1 and x_2, where "greatest" means greatest with respect to the divides relation $\preceq_{div}$. This property does not suggest a computational method, but it does describe the behavior we expect from the greatest-common-divisor function.

Although both of these are sentences in the theory of the nonnegative integers, we regard the former as a *program*, because it describes a method of computation, and the latter as a *specification*, because it describes the intended behavior of the program. Up to now, we have defined our functions by programs and then proved that they satisfy certain specifications.

In this chapter, we would like to reverse this procedure. We now suppose that we are given only a specification and try to derive a program that satisfies the specification. Thus we might be given the *greatest-common-divisor* property as a specification and attempt to derive the computationally suggestive axiom as a program.

As another example ⟪ out? ⟫, to specify the *quotient-remainder* program in the theory of the nonnegative integers, we might be given the *quotient-remainder* property

$$\textit{if}\ \ \textit{not}\ (x_2 = 0)\ \ \textit{then}\ \begin{bmatrix} x_1 = x_2 \cdot z_1 + z_2 \\ \quad \textit{and} \\ z_2 < x_2 \end{bmatrix}.$$

Here z_1 is the quotient and z_2 the remainder of dividing x_1 by x_2. The sentence specifies the behavior of the program only for the case in which the divisor x_2 is not zero; otherwise we do not care what the program returns.

From this specification we may hope to derive a program such as ⟪ less space around minus ⟫

$$quot(x_1,\ x_2) = \begin{cases} \textit{if}\ \ x_1 < x_2 \\ \textit{then}\ \ 0 \\ \textit{else}\ \ quot(x_1 - x_2,\ x_2) + 1 \end{cases}$$

$$\textit{and}$$

$$rem(x_1,\ x_2) = \begin{cases} \textit{if}\ \ x_1 < x_2 \\ \textit{then}\ \ x_1 \\ \textit{else}\ \ rem(x_1 - x_2,\ x_2). \end{cases}$$

⟪ mention program transformation here?? ⟫

So far, we have used the deductive tableau system only to establish the validity of sentences. In this chapter we extend it to derive programs from specifications as well.

14.1 SPECIFICATIONS AND PROGRAMS

In this section we are a little more precise about the specifications we accept, the programs we derive, and the relationship between them.

In a theory, we suppose that we are given a *specification* sentence

$$Q[\overline{x}; \overline{z}],$$

$\overline{x}$ is an abbreviation for $x_1, x_2, \ldots, x_m$, the *input variables* and $\overline{y}$ is an abbreviation for $y_1, y_2, \ldots, y_n$, the *output variables*. (We use the semicolon informally, instead of a comma, to separate the input and output variables.) It is assumed that there are no free variables in Q other than $\overline{x}$ and $\overline{z}$.

For example, the specification for the *quotient-remainder* program is the sentence

$$Q[x_1, x_2; z_1]: \quad \begin{array}{l} integer(z_1) \\ \quad and \\ z_1 \preceq_{div} x_1 \ \ and \ \ z_1 \preceq_{div} x_2 \\ \quad and \\ (\forall\ integer\ y) \left[\begin{array}{l} if \ \ y \preceq_{div} x_1 \ \ and \ \ y \preceq_{div} x_2 \\ then \ \ y \preceq_{div} z_1 \end{array}\right] \end{array}$$

We are also given *input sorts* $\overline{obj}(\overline{x})$, that is,

$$obj_1(x_1),\ obj_2(x_2),\ \ldots,\ obj_m(x_m),$$

where each obj_i is a unary predicate symbol, which characterizes a class of elements in the theory. We would like to define functions $\overline{f}(\overline{x})$, that is, $f_1(\overline{x})$, $f_2(\overline{x})$, $\ldots$, and $f_n(\overline{x})$. Each function f_i is intended to be applied to *inputs* $\overline{x} = x_1, x_2, \ldots, x_m$, where each *input* x_i is an element of the class obj_i. We assume that the function symbols $\overline{f}$ are "new," in the sense that they do not occur so far in the vocabulary of the theory. For example, for the *quotient-remainder* program, obj_1 and obj_2 are both *integer*, and f_1 is *gcd*.

We define the functions $\overline{f}$ by constructing a *program* $P[\overline{x}] : \overline{f}(\overline{x}) = \overline{t}[\overline{x}]$, that is, a sentence

$$P[x]: \quad \begin{array}{l} f_1(\overline{x}) = t_1[\overline{x}] \ \textit{and} \\ f_2(\overline{x}) = t_2[\overline{x}] \ \textit{and} \\ \quad \vdots \\ f_n(\overline{x}) = t_n[\overline{x}], \end{array}$$

where each $t_j[\overline{x}]$ is a term containing no variables other than $\overline{x}$.

For example, the *greatest-common-divisor* program we shall define is

$$P[x_1, x_2]: \quad gcd(x_1, x_2) \ = \ \begin{cases} \textit{if} \ x_2 = 0 \\ \textit{then} \ x_1 \\ \textit{else} \ gcd\big(x_2, \ rem(x_1, x_2)\big). \end{cases}$$

The program must *satisfy* the specification, in the sense that the *correctness* condition

$$\begin{array}{l} \textit{if} \ \ (\forall \ \overline{obj} \ \overline{x}) P[\overline{x}] \\ \textit{then} \ \ (\forall \ \overline{obj} \ \overline{x}) Q\big[\overline{x}; \ \overline{f}(\overline{x})\big] \end{array}$$

must be valid in the theory. Here $(\forall \ \overline{obj} \ \overline{x})$ is an abbreviation for

$$(\forall \ obj_1 \ x_1)(\forall \ obj_2 \ x_2) \cdots (\forall \ obj_m \ x_m).$$

For example, the *greatest-common divisor* program satisfies the specification in the sense that the correctness condition

$$\begin{array}{l} \textit{if} \ \ (\forall \ \textit{integer} \ x_1, \ x_2) P[x_1, x_2] \\ \textit{then} \ \ (\forall \ \textit{integer} \ x_1, \ x_2) Q\big[x_1, x_2; \ gcd(x_1, x_2)\big] \end{array}$$

is valid in the theory of the nonnegative integers.

In any theory, there are some symbols that are in the basic vocabulary, that have been defined by axioms that suggest some method of computation, or that define functions for which programs have already been derived. These "primitive" symbols may be used freely in a program sentence. In the theory of the nonnegative integers, for example, the constant symbol 0 and the successor function symbol $x + 1$ are in the basic vocabulary. The multiplication function symbol $x \cdot y$ and the less-than predicate symbol $x < y$ have been defined by computationally suggestive axioms. Therefore, 0, $x + 1$, $x \cdot y$ and $x < y$ are primitive symbols we can use in the greatest-common-divisor program.

On the other hand, there are symbols that denote entities we do not know how to compute, such as quantifiers and skolem function symbols. These "non-primitive" symbols may occur in specifications and in the axioms for a theory but not in any program.

In short, to ensure that the programs $\overline{f}(\overline{x}) = \overline{t}[\overline{x}]$ we derive are actually computationally suggestive, we shall require that only primitive constant, function, and predicate symbols, including the function symbols $\overline{f}$ themselves, may occur in the terms $\overline{t}[\overline{x}]$. We shall define primitivity more precisely later.

Remark (single input or output)

In the case in which the program has only one input or only one output, we shall drop the subscript and write x or z instead of x_1 or z_1. Thus, in the above example we may write $Q[x_1, x_2; z]$ instead of $Q[x_1, x_2; z_1]$. ∎

Example (quotient-remainder) ⟪ out? ⟫

In the theory of the nonnegative integers, we may specify the *quotient-remainder* program by the sentence

$$Q[x_1, x_2; z_1, z_2]: \quad \textit{if } \textit{not } (x_2 = 0) \textit{ then } \begin{bmatrix} \textit{integer}(z_1) \textit{ and } \textit{integer}(z_2) \\ \textit{and} \\ x_1 = x_2 \cdot z_1 + z_2 \\ \textit{and} \\ z_2 < x_2 \end{bmatrix},$$

where the input sorts obj_1 and obj_2 are both *integer*. Here z_1 is the quotient and z_2 is the remainder of dividing x_1 by x_2. The sentence specifies the behavior of the program only for the case in which the divisor x_2 is not zero; otherwise we do not care what the program returns. ⟪ out?? ⟫

Using the extended deductive-tableau system, we shall be able to derive the program

$$P[x_1, x_2]: \quad \begin{array}{l} quot(x_1, x_2) = \begin{cases} \textit{if } x_1 < x_2 \\ \textit{then } 0 \\ \textit{else } quot(x_1 - x_2, x_2) + 1 \end{cases} \\ \quad \textit{and} \\ rem(x_1, x_2) = \begin{cases} \textit{if } x_1 < x_2 \\ \textit{then } x_1 \\ \textit{else } rem(x_1 - x_2, x_2) \end{cases} \end{array}$$

⟪ less space around minus?? ⟫

The correctness condition in this case is the sentence

$$\begin{aligned} &\textit{if } (\forall \textit{ integer } x_1, x_2) P[x_1, x_2] \\ &\textit{then } (\forall \textit{ integer } x_1, x_2) Q[x_1, x_2; \textit{quot}(x_1, x_2), \textit{rem}(x_1, x_2)], \end{aligned}$$

which can be shown to be valid in the theory of the nonnegative integers. Thus the program does satisfy the specification. It also contains only primitive symbols. ∎

Example (redhead)

This example suggests that deriving programs may have applications to "database retrieval." We outline a new family theory. In the intended interpretation for this theory,

$\textit{person}(x)$ is the person relation ("x is a person")

$\textit{par}(x, y)$ is the parent relation ("x is a parent of y")

$\textit{red}(x)$ is the redheadedness relation ("x is redheaded").

One of the axioms for the theory is

$$(\forall \textit{ person } u, v) \begin{bmatrix} \textit{if } \textit{par}(u, v) \\ \textit{then } \textit{not par}(v, u) \end{bmatrix},$$

that is, the parent relation is asymmetric.

We are given the specification sentence

$$\begin{aligned} Q[x_1, x_2, x_3; z_1, z_2]: \quad &\textit{if } \textit{par}(x_1, x_2) \textit{ and } \textit{par}(x_2, x_3) \textit{ and} \\ &\quad \textit{red}(x_1) \textit{ and } \textit{not red}(x_3) \\ &\textit{then } \textit{person}(z_1) \textit{ and } \textit{person}(z_2) \textit{ and} \\ &\quad \textit{par}(z_1, z_2) \textit{ and} \\ &\quad \textit{red}(z_1) \textit{ and } \textit{not red}(z_2), \end{aligned}$$

where the input sorts obj_1, obj_2, and obj_3 are all *person*. In other words, if x_1 is a grandparent of x_3, and x_1 is redheaded but x_3 is not, then we are to find people z_1 and z_2 such that z_1 is a parent of z_2 and z_1 is redheaded but z_2 is not. We regard all three predicate symbols as primitive.

The *redhead* program we are to derive from this specification is

$$rh(x_1, x_2, x_3) = \begin{cases} \textit{if } \textit{red}(x_2) \\ \textit{then } x_2 \\ \textit{else } x_1 \end{cases}$$

$P[x_1, x_2, x_3]$: *and*

$$nrh(x_1, x_2, x_3) = \begin{cases} \textit{if } \textit{red}(x_2) \\ \textit{then } x_3 \\ \textit{else } x_2 \end{cases}$$

This program can be shown to satisfy the specification, in the sense that

if $(\forall\ \textit{person}\ x_1, x_2, x_3)P[x_1, x_2, x_3]$
then $(\forall\ \textit{person}\ x_1, x_2, x_3)Q[x_1, x_2, x_3; rh(x_1, x_2, x_3), nrh(x_1, x_2, x_3)]$

is valid in the theory. ∎

The derivation of this program will be given later in the chapter.

PROGRAM TRANSFORMATION

Up to now, we have been considering specifications that describe a relation between the inputs and outputs but do not suggest any method of computation. Sometimes, however, we know a method for computing the function and want to find another, perhaps more efficient one. This is known as *program transformation*. The same extended deductive-tableau system we use for ordinary program derivation will also be used for program transformation.

Example (flattree)

In a combined theory of trees and strings (Section [I]8.4), we introduced a function $\textit{flattree}(x)$ to form a string from the atoms of a given tree x. The function was defined by the following pair of computationally suggestive axioms

$$(\forall\ \textit{atom}\ u)[\textit{flattree}(u) = u] \qquad (\textit{atom})$$

$$(\forall\ \textit{tree}\ u, v)[\textit{flattree}(u \circ v) = \textit{flattree}(u) * \textit{flattree}(f)] \qquad (\textit{construction})$$

Suppose we would like to discover a different method for computing the same function. Then we may attempt to derive a program $\textit{flattree}1(x)$, whose specification is simply

$$Q1[x; z]: \ z = \textit{flattree}(x)_1.$$

Our input sort is *tree*.

We shall be able to derive many different programs to meet the above specification, all of them computing the same *flattree* function. Some of these programs will use the computational method suggested by the *flattree* axioms; others will use different, perhaps more efficient, methods.

The derivation of the *flattree*1 program is facilitated if we first derive a program $flattree2(x_1, x_2)$ to form a string from the atoms of a given tree x_1 and concatenate that string and a given string x_2. The specification sentence for *flattree*2 is

$$Q_2[x_1, x_2; z]: \quad z = flattree(x_1) * x_2,$$

Our input sorts obj_1 and obj_2 are *tree* and *string* respectively.

From this specification, we can construct a program sentence such as

$$P_2[x_1, x_2]: \quad flattree2(x_1, x_2) = \begin{cases} if \ \ atom(x_1) \\ then \ \ x_1 * x_2 \\ else \ \ flattree2\big(left(x_1), \\ \qquad flattree2\big(right(x_1), x_2\big)\big) \end{cases}$$

This program satisfies the specification, in the sense that the *correctness* condition

$\langle\!\langle$ out $\rangle\!\rangle$

$$if \ \ \begin{matrix}(\forall\ tree\ x_1)\\(\forall\ string\ x_2)\end{matrix}\Big[P[x_1, x_2] if \ \ atom(x_1)\Big]$$

$$then \ \ \begin{matrix}(\forall\ tree\ x_1)\\(\forall\ string\ x_2)\end{matrix}\Big[flattree2(x_1, x_2) = flattree(x_1) * x_2\Big]$$

is valid.

Once we have derived the program for *flattree*2, we may use it in deriving a program for *flattree*1. The program we obtain is

$$P1: \ flattree1(x) = flattree2(x, \Lambda).$$

The computational method described by the *flattree*1 and *flattree*2 programs turns out to be more efficient than that suggested by the original axioms for *flattree*.

This derivation will be presented in full detail later in the chapter. ∎

THE APPROACH

Suppose that we are given a specification

$$Q[\overline{x}; \overline{z}]$$

with input sorts $\overline{obj}$, and would like to derive a program

$$\overline{f}(\overline{x}) = \overline{t}[\overline{x}]$$

that satisfies this specification. We shall extend the deductive-tableau system so that the program can be obtained as a byproduct of proving the sentence

$$(\dagger) \qquad (\forall\ \overline{obj}\ \overline{x})(\exists\ \overline{z})Q[\overline{x}; \overline{z}].$$

In other words, we prove the existence of output objects $\overline{z}$ satisfying the given specification for given input objects $\overline{x}$. The proof must indicate a method for finding the desired output objects, and this method provides the computational basis for the program $\overline{f}(\overline{x}) = \overline{t}[\overline{x}]$ that computes the output objects.

We shall now discuss how the deductive-tableau system can be extended so that programs can be extracted from proofs.

14.2 OUTPUT ENTRIES

Up to now, a deductive tableau has had two columns, one for assertions and one for goals. To derive a program, we now extend our tableaux by introducing a number of *output columns*.

If we are given a specification

$$Q[x_1, \ldots, x_m; z_1, \ldots, z_n]$$

with n output variables, we introduce n new output columns. The jth output column is used in the derivation of the jth conjunct

$$f_j(\overline{x}) = t_j[\overline{x}]$$

of the desired program. It is labelled $f_j(\overline{a})$, where $\overline{a} = a_1, a_2, \ldots, a_m$ are new constants called the *input constants*. For any row, whether it contains an assertion or a goal, the output columns may all be blank or may each contain a term of the theory, called an *output entry*. The output entries will be used to derive programs from specifications.

We shall call a tableau with output columns an *extended tableau*, in contrast to a tableau with only assertion and goal columns, which we shall henceforth call a *basic tableau*.

Example (extended tableau)

The following is an extended tableau from the derivation of a *quotient-remainder* program:

assertions	goals	$quot(a_1, a_2)$	$rem(a_1, a_2)$
$not\ (a_2 = 0)$			
	$integer(z_1)$ *and* $integer(z_2)$ *and* $a_1 = a_2 \cdot z_1 + z_2$ *and* $z_2 < a_2$	z_1	z_2
$u \cdot 0 = 0$			
	$a_1 < a_2$	0	a_1
$a_1 < a_2$		$quot(a_1 - a_2, a_2) + 1$	$rem(a_1 - a_2, a_2)$

— Tableau T_1 —

Here the assertions $not(a_2 = 0)$ and $u \cdot 0 = 0$ have no output entries; the other rows all do. ∎

SUITING A ROW

We have given the meaning of the assertions and goals of a basic tableau by defining its truth (under an interpretation) and validity (in a theory). The same definitions apply to extended tableaux, ignoring the output entries. To give the meaning of the output entries themselves, we define what it means for terms to *suit a tableau* (under an interpretation) and *satisfy a tableau* (in a theory). Loosely speaking, the terms that suit a tableau will denote acceptable outputs for the desired program. We first define what it means for terms to suit a single row of a tableau. The terms that suit the row will be acceptable outputs when that row's assertion is false [or that row's goal is true].

In the definition that follows, we consider a row of a tableau with an assertion A [or goal G] and output entries $\bar{s} = s_1, \ldots, s_n$, that is,

$a_1, a_2)$

assertions	goals	$\overline{f}(\overline{a})$
A		$\overline{s}$

or

	$\mathcal{G}$	$\overline{s}$

We consider any substitution λ such that the instances $A \lhd \lambda$ [or $\mathcal{G} \lhd \lambda$] and $\overline{s} \lhd \lambda$ are closed, that is, they contain no free variables. According to the definition, the terms $\overline{s} \lhd \lambda$ *suit the row* under an interpretation I if $A \lhd \lambda$ is false [or $\mathcal{G} \lhd \lambda$ is true] under I. But let us be more precise.

Definition (*terms suit row*)

Consider a row of a tableau containing an assertion A [or goal $\mathcal{G}$]. Let $\overline{t} = t_1, t_2, \ldots, t_n$ be closed terms and I be an interpretation. We shall say that the terms $\overline{t}$ *suit the row* under I if, for some substitution λ, the following conditions hold:

- *Truth condition.* The sentence $A \lhd \lambda$ is closed and false under I [or the sentence $\mathcal{G} \lhd \lambda$ is closed and true under I].
- *Output condition.* If the row has output entries $\overline{s} = s_1, s_2, \ldots, s_n$, the instances

  $$\overline{s} \lhd \lambda = s_1 \lhd \lambda, s_2 \lhd \lambda, \ldots, s_n \lhd \lambda$$

  are closed and have the same values, respectively, as $\overline{t} = t_1, t_2, \ldots, t_n$ under I.

We shall call such a substitution λ a *suiting* substitution. ⌟

Remark (rows with no output entries)

The *output* condition holds vacuously in the case in which the row has no output entries. In other words, any closed terms $\overline{t}$ will suit such a row, provided there is some substitution λ for which $A \lhd \lambda$ is false [or $\mathcal{G} \lhd \lambda$ is true] and closed under I.

When it is convenient, such a row may actually be treated as a row with output entries

$$\overline{u} = u_1, u_2, \ldots, u_n,$$

where the u_i are distinct variables that do not occur free elsewhere in the row. For if the closed terms $\bar{t}$ suit a row with output entries $\bar{u}$, then $\bar{t}$ suit the corresponding row with no output entries at all, with the same suiting substitution, because the *output* condition holds vacuously. And if $\bar{t}$ suit a row that has no output entries, with suiting substitution λ, then $\bar{t}$ suit the corresponding row with output entries $\bar{u}$, with suiting substitution

$$\{u_1 \leftarrow t_1,\ u_2 \leftarrow t_2,\ \ldots,\ u_n \leftarrow t_n\} \square \lambda.$$

Here $\square$ is the composition function for substitutions.

By the same token, we may find it convenient to treat a row whose output entries $\bar{u}$ are all distinct variables that do not occur free elsewhere in the row as a row without output entries at all. ⌟

Let us now illustrate the new definition.

Example (suiting a row)

The following ⟪ ?? above? ⟫ extended tableau T_2 is obtained from a derivation of the *redhead* program described in an earlier example.

Let I_r be any model for the theory under which

$$red(a_2) \text{ is true,}$$

that is, a_2 is a redheaded person, and under which assertion 1 is true, that is, $par(a_1, a_2)$, $par(a_2, a_3)$, $red(a_1)$, and *not* $red(a_3)$ are all true. In other words, a_1 is a parent of a_2, a_2 is a parent of a_3, and a_1 is redheaded but a_3 is not.

Under this interpretation, the two closed terms

$$\bar{t}: \quad a_2,\ a_3$$

suit row 5,

	5. $red(a_2)$	a_2	a_3

which has no free variables. To show this, we may take the suiting substitution λ to be the empty substitution { }. The *truth* condition holds under I_r, because the instance of the goal, $\mathcal{G} \cdot \lambda$, is

$$red(a_2),$$

which is true under I_r. (In this discussion, we shall use $\mathcal{A}$ or $\mathcal{G}$ to stand for the assertion or goal of the row under discussion.) The *output* condition also holds,

assertions	goals	$rh(a_1, a_2)$	$nrh(a_1, a_2)$
1. $par(a_1, a_2)$ *and* $par(a_2, a_3)$ *and* $red(a_1)$ *and* *not* $red(a_3)$			
	2. $par(z_1, z_2)$ *and* $red(z_1)$ *and* *not* $red(z_2)$	z_1	z_2
3. *if* $par(u, v)$ *then* *not* $par(v, u)$			
	4. *not* $red(a_2)$	a_1	a_2
	5. $red(a_2)$	a_2	a_3
	6. *true*	*if* $red(a_2)$ *then* a_2 *else* a_1	*if* $red(a_2)$ *then* a_3 *else* a_2

— Tableau T_2 —

because the corresponding instances of the output entries, $a_2 \blacktriangleleft \lambda$ and $a_3 \blacktriangleleft \lambda$, are precisely the same as the closed terms $\bar{t}: a_2, a_3$.

Under the same interpretation, the same closed terms $\bar{t}: a_2, a_3$ also suit row 2,

	2. $par(z_1, z_2)$ *and* $red(z_1)$ *and* *not* $red(z_2)$	z_1	z_2

which does have free variables. To show this, we take the suiting substitution λ to be

$$\{z_1 \leftarrow a_2,\ z_2 \leftarrow a_3\}.$$

The *truth* condition holds under I_r, because the instance of the goal, $\mathcal{G} \blacktriangleleft \lambda$, is

$$par(a_2, a_3) \text{ and } red(a_2) \text{ and not } red(a_3),$$

which is true under I_r. The *output* condition holds, because the corresponding instances of the output entries, $z_1 \blacktriangleleft \lambda$ and $z_2 \blacktriangleleft \lambda$, are precisely the same as the closed terms $\bar{t}: a_2, a_3$.

The same closed terms $\bar{t}$: a_2, a_3 also suit row 6,

	6. *true*	*if* $red(a_2)$ *then* a_2 *else* a_1	*if* $red(a_2)$ *then* a_3 *else* a_2

under I_r. To show this, we take the suiting substitution to be the empty substitution { }. The *truth* condition of course holds, because any instance of the goal *true* is true under I_r. The *output* condition also holds; although the corresponding instances of the output entries,

if $red(a_2)$ / *then* a_2 / *else* a_1 and *if* $red(a_2)$ / *then* a_3 / *else* a_2,

are not identical to $\bar{t}$: a_2, a_3, they have the same values under I_r, because $red(a_2)$ is true under I_r.

By the same reasoning, the closed terms

$\bar{t}$: *if* $red(a_2)$ / *then* a_2 / *else* a_1 and *if* $red(a_2)$ / *then* a_3 / *else* a_2,

also suit row 6 under I_r, for these terms are identical to the output entries for this row. In fact, these terms also suit rows 2 and 5 under I_r.

⟪ remaining examples out?? ⟫

Let I_n be any model for the theory under which

$red(a_2)$ is false,

that is, a_2 is not a redheaded person, and under which, as in I_r, assertion 1 is true, that is, $par(a_1, a_2)$, $par(a_2, a_3)$, $red(a_1)$, and *not* $red(a_3)$ are all true. Under this interpretation, the two closed terms

$\bar{t}$: a_1, a_2

suit rows 4 and 2

	4. *not* $red(a_2)$	a_1	a_2
	2. $par(z_1, z_2)$ *and* $red(z_1)$ *and not* $red(z_2)$	z_1	z_2

To show this ⟪ out?? ⟫, we take the suiting substitution λ to be { } and

$\{z_1 \leftarrow a_1, z_2 \leftarrow a_2\}$,

respectively. The same terms $\bar{t}: a_1, a_2$ can be shown to suit row 6

	6. *true*	*if* $red(a_2)$ *then* a_2 *else* a_1	*if* $red(a_2)$ *then* a_3 *else* a_2

under I_n.

We can also show that the terms

$$\bar{t}: \begin{array}{l} \textit{if } red(a_2) \\ \textit{then } a_2 \\ \textit{else } a_1 \end{array} \quad \text{and} \quad \begin{array}{l} \textit{if } red(a_2) \\ \textit{then } a_3 \\ \textit{else } a_2, \end{array}$$

suit rows 2, 4, and 6 under I_n. In fact, these terms have the same value as a_1 and a_2 under I_n, because $red(a_2)$ is false.

Let I_0 be any model for the theory under which assertion 1,

1. $par(a_1, a_2)$ *and* $par(a_2, a_3)$ *and* $red(a_1)$ *and not* $red(a_3)$			

is false. Then any closed terms $\bar{t}$ suit this row under I_0. To show this, we take the suiting substitution to be $\{\ \}$. The *truth* condition holds under I_0, because the instance of the assertion, $A \leftarrow \lambda$, is A itself, which is closed and false under I_0. The output condition holds vacuously, because this row has no output entries.

Suppose I is any interpretation under which some instance

$$\begin{bmatrix} \textit{if } par(u, v) \\ \textit{then not } par(v, u) \end{bmatrix} \leftarrow \lambda$$

of the *asymmetry* axiom in row 3,

3. *if* $par(u, v)$ *then not* $par(v, u)$		

for the parent relation is false. (Consequently I is not a model for the theory.) Then any closed terms $\bar{t}$ suit row 3, under I. Taking the suiting substitution to be λ itself, we see that the *truth* condition holds, because $A \leftarrow \lambda$ has been assumed to be false under I. The *output* condition holds vacuously, because the row has no output entries. ⟪ explain purpose of examples? ⟫ ∎

Remark (true assertions)

Suppose I is any interpretation under which no closed instance $A \leftarrow \lambda$ of the assertion A is false [or no closed instance of the goal G is true]. Then no closed terms $\bar{t}$ suit the assertion A [or the goal G] under I because the *truth* condition cannot hold. In particular, if A is valid in a theory, no closed terms $\bar{t}$ can suit the assertion A under a model for the theory. ∎

SUITING A TABLEAU

We can now say what it means for terms to suit an entire tableau, rather than a single row, under an interpretation.

Definition (terms suit tableau)

Let T be a tableau, $\bar{t} = t_1, t_2, \ldots, t_n$ be closed terms, and I be an interpretation. We shall say that the terms $\bar{t}$ *suit the tableau* T under I if $\bar{t}$ suit some row of T under I. ∎

Example (suiting a tableau) ⟪ **Marianne: too many examples** ⟫

Let us refer back to the tableau T_2 of the preceding example, from the derivation of the *redhead* program.

We have seen that the closed terms

$$\bar{t}: a_2, a_3$$

suit the rows 2, 5, and 6 under the interpretation I_r, in which a_2 is redheaded. Therefore, these terms suit the entire tableau under I_r.

Also, the terms

$$\bar{t}: a_1, a_2$$

suit the rows 2, 4, and 6 under the interpretation I_n, in which a_2 is not redheaded. Therefore, these terms suit the entire tableau under I_n.

It can be shown that a_2 and a_3 do not suit the tableau under I_n, that is, they do not suit any row under this interpretation. Similarly, a_1 and a_2 do not suit the tableau under I_r.

We have also seen that the terms

$$\bar{t}: \quad \begin{array}{l} \textit{if } \textit{red}(a_2) \\ \textit{then } a_2 \\ \textit{else } a_1 \end{array} \quad \text{and} \quad \begin{array}{l} \textit{if } \textit{red}(a_2) \\ \textit{then } a_3 \\ \textit{else } a_2 \end{array}$$

suit the rows 2, 5, and 6 under the interpretation I_r. Therefore, these terms suit the *entire* tableau under I_r. These same terms suit the rows 2, 4, and 6 under I_n; therefore, they suit the entire tableau under I_n. ∎

SATISFYING A TABLEAU

Finally, we can say what it means for *terms* to *satisfy* a tableau for a given theory. By discovering closed terms that satisfy the appropriate tableau, we shall be able to construct a program that satisfies a given specification.

Definition (terms satisfy tableau)

Let T be a tableau, and $\bar{t} = t_1, t_2, \ldots, t_n$ be closed terms.
In a given theory, the terms $\bar{t}$ ***satisfy the tableau*** T if $\bar{t}$ suit T under every model for the theory. ∎

Example (satisfying a tableau)

Consider once more the tableau T_2 from the derivation of the *redhead* program. We claim that, in the family theory described earlier, the terms

$$\bar{t}: \quad \begin{array}{l} \textit{if } \textit{red}(a_2) \\ \textit{then } a_2 \\ \textit{else } a_1 \end{array} \quad \text{and} \quad \begin{array}{l} \textit{if } \textit{red}(a_2) \\ \textit{then } a_3 \\ \textit{else } a_2 \end{array}$$

satisfy this tableau. For consider any model I for the theory; we show that $\bar{t}$ suit the tableau under I. We distinguish among three cases.

- If

  $$\textit{red}(a_2)$$

  and

  $$\textit{par}(a_1, a_2) \textit{ and } \textit{par}(a_2, a_3) \textit{ and } \textit{red}(a_1) \textit{ and not } \textit{red}(a_3)$$

are both true under I, we have seen that the terms $\bar{t}$ suit the rows 2, 5, and 6. Since they suit a least one row, they suit the entire tableau.

- If

  $$not\ red(a_2)$$

  and

  $$par(a_1, a_2)\ and\ par(a_2, a_3)\ and\ red(a_1)\ and\ not\ red(a_3)$$

  are true under I, we have seen that the terms $\bar{t}$ suit the rows 2, 4, and 6, and hence the entire tableau.

- Finally, if

  $$par(a_1, a_2)\ and\ par(a_2, a_3)\ and\ red(a_1)\ and\ not\ red(a_3)$$

  is false under I, we have seen that any terms will suit row 1, and hence the entire tableau.

These three cases exhaust all possibilities. Therefore, $\bar{t}$ suit the tableau under I, as we wanted to show. ∎

14.3 PROPERTIES OF EXTENDED TABLEAUX

The properties we have established for basic deductive tableaux carry over to extended tableaux. In particular, the duality, instantiation, and renaming properties of basic tableaux all have their counterparts for tableaux with output entries. We begin by adapting the notion of equivalence to extended tableaux.

Definition (equivalence)

In a theory, two tableaux T and T' are *equivalent* if, for any model I for the theory,

T is true under I
precisely when
T' is true under I

and, for any closed terms $\bar{t}$,

$\bar{t}$ suit $\mathcal{T}$ under $\mathcal{I}$
precisely when
$\bar{t}$ suit $\mathcal{T}'$ under $\mathcal{I}$. ⌟

Sometimes the notion of equivalence is too strong. We introduce a weaker notion, that of two tableaux having the "same meaning."

Definition (same meaning) ⟪ **out?** ⟫ ⟪ **later?** ⟫ ⟪ **example?** ⟫

In a theory, two tableaux $\mathcal{T}$ and $\mathcal{T}'$ have the *same meaning* if

$\mathcal{T}$ is valid in the theory
precisely when
$\mathcal{T}'$ is valid in the theory

and, for any closed terms $\bar{t}$,

$\bar{t}$ satisfy $\mathcal{T}$ in the theory
precisely when
$\bar{t}$ satisfy $\mathcal{T}'$ in the theory. ⌟

It is clear that if two tableaux are equivalent, they have the same meaning.

We can now state the three properties as they are adapted for extended tableaux.

DUALITY

The *duality* property states that we can move sentences freely between the assertion and goal columns simply by negating them, obtaining an equivalent tableau.

Property (duality)

In a theory,

- A tableau containing an assertion A with output entries $\bar{s}$ [or none]

  is equivalent to

the tableau containing instead the goal (*not* A) with the same output entries $\bar{s}$ [or none].

- A tableau containing a goal G with output entries $\bar{s}$ [or none]

  is equivalent to

  the tableau containing instead the assertion (*not* G) with the same output entries $\bar{s}$ [or none]. ∎

The justification is straightforward, but we present it to give the definitions some exercise.

Justification (duality)

We show only the first part. Let I be any model for the theory in question. Let T be the tableau with the assertion A and output entries $\bar{s}$ [if any], and T' be the tableau with the goal (*not* A) and output entries $\bar{s}$ [if any] instead. By the *duality* property for basic tableaux, we know that T is true under I if and only if T' is true under I.

Suppose that the closed terms $\bar{t}$ suit T under I; then they suit some row of T under I. If that row is not that of the assertion A, then the row also occurs in T', so the terms $\bar{t}$ also suit T' under I.

In the case in which $\bar{t}$ suit the assertion A itself under I, we know that there is a suiting substitution λ such that the *truth* condition holds, that is,

$A \blacktriangleleft \lambda$ is closed and false under I,

and the *output* condition holds, that is,

(†) the instances $\bar{s} \blacktriangleleft \lambda$ of the output entries [if any] are closed and have the same values, respectively, as $\bar{t}$ under I.

From the *truth* condition, we have

(‡) (*not* A)$\blacktriangleleft \lambda$ is closed and true under I.

Hence (by (†) and (‡)) the terms $\bar{t}$ suit the goal (*not* A) in the tableau T', with suiting substitution λ, and therefore suit the tableau T' itself.

Similarly, we can show that, if closed terms $\bar{t}$ suit T' under I, they also suit T under I. Hence T and T' are equivalent. ∎

RENAMING

The *renaming* property states that we may systematically rename the free variables of any row, obtaining an equivalent tableau. Recall that a ***permutation*** substitution is one that always replaces distinct variables with distinct variables ⟪ Section 6.8?? ⟫. Permutation substitutions have ***inverses***; in fact, π is a permutation substitution if and only if there is a permutation substitution π^{-1} such that

$$\pi \square \pi^{-1} = \{\ \}.$$

Property (renaming)

In a theory, for any permutation substitution π,

a tableau containing an assertion [or goal] $\mathcal{F}$ with output entries $\bar{s}$ [or none]

is equivalent to

the tableau containing instead the assertion [or goal] $\mathcal{F} \triangleleft \pi$ with the output entries $\bar{s} \triangleleft \pi$ [or none]. ∎

Justification (renaming)

Let I be any model for the theory in question, and let π be any permutation substitution. We consider only the assertion case. Let $\mathcal{T}$ be the tableau with the assertion $\mathcal{F}$ and output entries $\bar{s}$ [or none], and $\mathcal{T}'$ be the tableau with the assertion $\mathcal{F} \triangleleft \pi$ and output entries $\bar{s} \triangleleft \pi$ [or none] instead. By the *renaming* property for basic tableaux, we know that $\mathcal{T}$ is true under I if and only if $\mathcal{T}'$ is true under I.

Suppose the closed terms $\bar{t}$ suit $\mathcal{T}$ under I. Then they suit some row of $\mathcal{T}$ under I. If that row is not the assertion $\mathcal{F}$, then the row also occurs in $\mathcal{T}'$, so the terms $\bar{t}$ also suit $\mathcal{T}'$ under I.

Suppose the terms $\bar{t}$ suit the assertion $\mathcal{F}$ under I. Then for some suiting substitution λ, we have the *truth* condition,

$\mathcal{F} \triangleleft \lambda$ is closed and false under I,

and the *output* condition,

the instances $\bar{s} \triangleleft \lambda$ of the output entries [if any] are closed and have the same values, respectively, as $\bar{t}$ under I.

Suppose π^{-1} is the inverse of π. Then $\pi \square \pi^{-1} = \{\ \}$ ⟪ out?? ⟫ ⟪ mentioned earlier? ⟫, and we have (by properties of substitutions)

$$\mathcal{F} \lhd \lambda \equiv (\mathcal{F} \lhd \{\ \}) \lhd \lambda \equiv (\mathcal{F} \lhd (\pi \square \pi^{-1})) \lhd \lambda \equiv ((\mathcal{F} \lhd \pi) \lhd \pi^{-1}) \lhd \lambda$$
$$\equiv (\mathcal{F} \lhd \pi) \lhd (\pi^{-1} \square \lambda)$$

and, if there are output entries $\bar{s}$,

$$\bar{s} \lhd \lambda = (\bar{s} \lhd \{\ \}) \lhd \lambda = (\bar{s} \lhd (\pi \square \pi^{-1})) \lhd \lambda = ((\bar{s} \lhd \pi) \lhd \pi^{-1}) \lhd \lambda$$
$$= (\bar{s} \lhd \pi) \lhd (\pi^{-1} \square \lambda).$$

Consequently (by the *truth* and *output* conditions),

$(\mathcal{F} \lhd \pi) \lhd (\pi^{-1} \square \lambda)$ is closed and false under I

and

$(\bar{s} \lhd \pi) \lhd (\pi^{-1} \square \lambda)$ are closed and have the same values, respectively, as $\bar{t}$ under I.

In other words, the terms $\bar{t}$ suit the assertion $\mathcal{F} \lhd \pi$, with suiting substitution $\pi^{-1} \square \lambda$, and therefore suit the tableau itself.

Similarly, we can show that, if closed terms $\bar{t}$ suit $\mathcal{T}'$ under I, they also suit $\mathcal{T}$ under I. ⟪ exercise?? ⟫. Hence $\mathcal{T}$ and $\mathcal{T}'$ are equivalent. ⌟

INSTANTIATION

The *instantiation* property states that we may add to the tableau any instance of any of its rows, obtaining an equivalent tableau.

Property (instantiation)

In a theory, for any substitution θ,

a tableau containing an assertion [or goal] $\mathcal{F}$ with output entries $\bar{s}$ [or none]

is equivalent to

the tableau containing in addition the assertion [or goal] $\mathcal{F} \lhd \theta$ with the output entries $\bar{s} \lhd \theta$ [or none]. ⌟

Note that, in the ***duality*** and ***renaming*** properties, we replaced one row with another; in this ***instantiation*** property, we add a new row but do not remove the original one. ⟪ The justification is left as an exercise? ⟫

Justification (instantiation)

Let I be any model for the theory in question, and let θ be any substitution. We consider only the assertion case.

Let T be a tableau with the assertion $\mathcal{F}$ and output entries $\bar{s}$ [or none], and T' be the tableau with the assertion $\mathcal{F} \triangleleft \theta$ and output entries $\bar{s} \triangleleft \theta$ [or none] in addition. By the ***instantiation*** property for basic tableaux, we know that T is true under I if and only if T' is true under I.

Suppose that the closed terms $\bar{t}$ suit T under I. Since every row in T is also in T', we know that $\bar{t}$ also suit T' under I.

Suppose, on the other hand, that the closed terms $\bar{t}$ suit T' under I. Then they suit some row of T' under I. If that row is not the assertion $\mathcal{F} \triangleleft \theta$, then the row also occurs in T, so the terms $\bar{t}$ also suit T under I.

Suppose the terms $\bar{t}$ suit the assertion $\mathcal{F} \triangleleft \theta$ in T' under I. Then, for some suiting substitution λ, we have the ***truth*** condition,

$$(\mathcal{F} \triangleleft \theta) \triangleleft \lambda, \text{ that is, } \mathcal{F} \triangleleft (\theta \square \lambda), \text{ is closed and false under } I,$$

and the ***output*** condition,

> the terms $(\bar{s} \triangleleft \theta) \triangleleft \lambda \;=\; \bar{s} \triangleleft (\theta \square \lambda)$ are closed and have the same values, respectively, as $\bar{t}$ under I.

Hence the terms $\bar{t}$ suit the assertion $\mathcal{F}$ in T, with suiting substitution $\theta \square \lambda$, and therefore suit the tableau T itself.

Hence T and T' are equivalent. ∎

VALID ASSERTION

The ***valid-assertion*** property states that we may add any valid assertion to a tableau, obtaining an equivalent tableau.

Property (valid assertion)

In any theory, for any valid sentence A,

a tableau T
is equivalent to
the tableau T' obtained from T by adding the assertion A. ∎

Justification (valid assertion)

⟪ easy? exercise ⟫

Let I be any model for the theory in question. That T is true under I precisely when T' is true under I follows from properties of basic tableaux.

Suppose the closed terms $\bar{t}$ suit T under I. Since every row of T is also a row of T', we know that $\bar{t}$ also suit T' under I.

Suppose, on the other hand, that the closed terms $\bar{t}$ suit T' under I. Then they suit some row of T' under I. That row cannot be the assertion A, because, as mentioned in a previous remark, no terms can suit a valid assertion of the theory; therefore $\bar{t}$ must suit one of the original rows of T. That is, $\bar{t}$ suit T under I. ∎

OUTPUT ENTRY

The *output entry* property is the one we use to relate the output entries of tableaux with the specifications of programs.

Property (output entry)

In any theory, if the closed terms $\bar{t}[\bar{a}]$ satisfy a tableau

assertions	goals	$\overline{f}(\bar{a})$
	$R[\bar{a}; \bar{z}]$	$\bar{z}$

where $\bar{a}$ are new constants and $\bar{z}$ are the only free variables in $R[\bar{a}; \bar{z}]$, then the sentence

$$(\forall \bar{z}) R[\bar{z}; \bar{t}[\bar{z}]]$$

is valid in the theory. ∎

Let us illustrate the property.

Example (output entry property)

In the theory of trees, suppose we establish that the closed term $a_1 \cdot a_2$ suits the tableau

assertions	goals	$\ell r(a_1, a_2)$
	$a_1 = \mathit{left}(z)$ *and* $a_2 = \mathit{right}(z)$	z

Then, by the *output entry* property, we know that the sentence

$$(\forall x_1, x_2)[x_1 = \mathit{left}(x_1 \cdot x_2) \text{ and } x_2 = \mathit{right}(x_1 \cdot x_2)]$$

is valid in the theory of trees. ∎

Justification (output entry)

Suppose that the closed terms $\bar{t}[\bar{a}]$ satisfy the above tableau. To show that $(\forall \bar{x})\mathcal{R}[\bar{x}; \bar{t}[\bar{x}]]$ is valid (in the theory), it suffices (by the *universal quantifier-elimination* proposition) to show that

$$\mathcal{R}[\bar{a}; \bar{t}[\bar{a}]]$$

is valid, since $\bar{a}$ are new constants. Consider an arbitrary model I; we show that

(*) $\mathcal{R}[\bar{a}; \bar{t}[\bar{a}]]$ is true under I.

Because $\bar{t}[\bar{a}]$ satisfy the above tableau, they suit the sole row of the tableau under I. In other words, for some suiting substitution λ, we have the *truth* condition,

$\mathcal{R}[\bar{a}; \bar{z}] \blacktriangleleft \lambda$ is closed and true under I,

and the *output* condition

$\bar{z} \blacktriangleleft \lambda$ are closed and have the same values, respectively,
as $\bar{t}[\bar{a}]$ under I.

Because $\bar{z}$ are all the free variables of $\mathcal{R}[\bar{a}; \bar{z}]$, this means that

$\mathcal{R}[\bar{a}; \bar{z} \blacktriangleleft \lambda]$ is true under I,

or, equivalently (by the *output* condition),

$$\mathcal{R}[\bar{a}; \bar{t}[\bar{a}]] \text{ is true under } \mathcal{I}.$$

But this is the condition (*) we wanted to show. ∎

⟪ example ⟫

Now that we have established the properties of extended deductive tableaux, we are ready to show how tableaux may be used for program derivation.

14.4 THE DERIVATION PROCESS

Let us review the problem of program derivation. We are given a specification

$$\mathcal{Q}[\bar{x}; \bar{z}]$$

in a theory, with input sorts $\overline{obj}$, and we would like to derive a program

$$\bar{f}(\bar{x}) \Leftarrow \bar{t}[\bar{x}]$$

that satisfies this specification. In other words, we want to find terms $\bar{t}[\bar{x}]$ such that the *correctness* condition

$$\begin{array}{l} \textit{if } (\forall\ \overline{obj}\ \bar{x})\left[\bar{f}(\bar{x}) = \bar{t}[\bar{x}]\right] \\ \textit{then } (\forall\ \overline{obj}\ \bar{x})\mathcal{Q}[\bar{x}; \bar{f}(\bar{x})] \end{array}$$

is valid in the theory.

THE INITIAL TABLEAU

Our approach is to prove the theorem

$$(\forall\ \overline{obj}\ \bar{x})(\exists\ \bar{z})\mathcal{Q}[\bar{x}; \bar{z}]$$

and obtain the terms $\bar{t}[\bar{x}]$ as a byproduct of the proof process.

This sentence is an abbreviation (using the relative quantifier notation) for

$$(\forall\ \bar{x})\left[\begin{array}{l} \textit{if } \overline{obj}(\bar{x}) \\ \textit{then } (\exists\ \bar{z})[\mathcal{Q}[\bar{x}; \bar{z}]] \end{array}\right]$$

We shall actually establish the skolemized form of this sentence,

$$\mathcal{Q}^*[\bar{a};\bar{z}]: \quad \begin{array}{l} \textit{if } \overline{obj}(\bar{a}) \\ \textit{then } \mathcal{Q}[\bar{a};\bar{z}], \end{array}$$

where $\bar{a} = a_1, a_2, \ldots, a_m$ are "new" constants, i.e., constants that do not already occur in the vocabulary of the theory.

At the same time, we shall derive the desired terms $\bar{t}[\bar{a}]$ using the output columns of the tableau. For this purpose, we establish the validity of a particular "initial tableau."

Definition (initial tableau)

For a given specification $\mathcal{Q}[\bar{x};\bar{z}]$ in a theory, with input sorts $\overline{obj}$, an *initial tableau* is as follows:

assertions	goals	$\bar{f}(\bar{a})$
	if $\overline{obj}(\bar{a})$ *then* $\mathcal{Q}[\bar{a};\bar{z}]$	$\bar{z}$

The skolem constants $\bar{a}$ are called the ***input constants***. Note that this tableau has n output columns, one for each output variable $\bar{z} = z_1, z_2, \ldots, z_n$.

An initial tableau may also contain as assertions any valid sentences of the theory in question, without output entries. ⌟

The initial tableau has an important property, expressed in the following result.

Proposition (initial tableau)

In any theory,

> if the closed terms $\bar{t}[\bar{a}]$ satisfy the initial tableau for a specification, then the program $\bar{f}(\bar{x}) = \bar{t}[\bar{x}]$ satisfies the specification. ⌟

Proof

Assume that, in a theory, the closed terms $\bar{t}[a]$ satisfy the initial tableau for the specification $Q[\bar{x};\bar{z}]$, with input sorts $\overline{obj}$. We show that the *correctness* condition

$$\begin{array}{l} \textit{if } (\forall\, \overline{obj}\ \bar{x})\left[\bar{f}(\bar{x}) = \bar{t}[\bar{x}]\right] \\ \textit{then } (\forall\, \overline{obj}\ \bar{x})Q[\bar{x},\, \bar{f}(\bar{x})] \end{array}$$

is valid in the theory.

Assume that (under a given model)

$$(*) \qquad (\forall\, \overline{obj}\ \bar{x})\left[\bar{f}(\bar{x}) = \bar{t}[\bar{x}]\right]$$

is true. We show that then

$$(\forall\, \overline{obj}\ \bar{x})Q\left[\bar{x};\bar{f}(\bar{x})\right]$$

is also true.

It suffices, by our assumption $(*)$, to show that

$$(\forall\, \overline{obj}\ \bar{x})Q\left[\bar{x};\bar{t}[\bar{x}]\right],$$

that is (expanding the relativized quantifier),

$$(\forall\, \bar{x})\left[\begin{array}{l} \textit{if } \overline{obj}(\bar{x}) \\ \textit{then } Q\left[\bar{x};\, \bar{t}(\bar{x})\right] \end{array}\right],$$

is true. By the ***output entry*** property, to show that the above sentence is actually valid, it suffices that the closed terms $\bar{t}[\bar{a}]$ satisfy the tableau

assertions	goals	$\bar{f}(\bar{a})$
	if $\overline{obj}(\bar{a})$ *then* $Q[\bar{a};\bar{z}]$	$\bar{z}$

The ***valid-assertion*** property allows us to add any valid sentences of the theory to this tableau as assertions, without changing the satisfying terms. In other words, it suffices that the closed terms $\bar{t}[\bar{a}]$ that satisfy the initial tableau, as we have assumed. ∎

Example (flattree1)

In our example of program transformation (Section ??), we specified the program *flattree*1, which computes the *flattree* function, with the sentence

$$Q^1[x; z] : \; z = \mathit{flattree}(x)$$

in a combined theory of trees and strings, where the input sort is *tree*.

The initial tableau is thus

assertions	goals	*flattree*1(*a*)
	if *tree*(*a*) *then* $z = \mathit{flattree}(a)$	z

Properties of the combined theory of trees and strings may be included in the initial tableau as assertions. ⌟

PRIMITIVE EXPRESSIONS

We have mentioned that some symbols are "primitive"; they denote objects, functions, or relations we know how to compute. Other symbols, such as quantifiers and some skolem functions, are "nonprimitive"; we do not know how to compute what they denote. We shall assume that at the beginning of each derivation we are given a list of primitive constant, function, and predicate symbols, called the *primitive list*. Typically this list shall include the truth symbols *true* and *false*, the propositional connectives, the term constructor *if-then-else*, the basic constant, function, and predicate symbols of the theory, and any symbols that have been defined by computationally suggestive axioms. In addition, we may include in the primitive list any function symbols for which we have already derived programs.

We may now define a *primitive expression* as follows:

We define an expression e to be *primitive* in a given initial tableau, if each symbol that occurs in e is a variable, an element of the primitive list, or one of the input constants $\overline{a}$. For example, for the *flattree*2 derivation, the term

$$\mathit{if} \;\; \mathit{atom}(a_1) \;\; \mathit{then} \;\; a_1 * a_2 \;\; \mathit{else} \;\; z$$

is primitive.

INTERMEDIATE TABLEAUX

When we are establishing the validity of a basic tableau, we apply deduction rules that add new rows but preserve validity, until we obtain the final assertion *false* or the final goal *true*. In deriving a program, we apply deduction rules to an extended tableau. In addition to maintaining the validity of the tableau, the extended rules will "maintain the satisfying terms" of the tableau. In other words, a closed term will satisfy the new tableau if and only if it satisfies the old tableau. When we obtain the final assertion *false* or the final goal *true*, its output entries will provide the final program, as we shall see.

The tableaux we develop all have a property expressed as follows.

Proposition (intermediate tableau)

In a theory, at each stage in the derivation of a program from a specification, the following property holds:

if the closed terms $\bar{t}[\bar{a}]$ satisfy the tableau,
then the program $\bar{f}(\bar{x}) = \bar{t}[\bar{x}]$ satisfies the specification. ⌟

Proof

To establish that the desired property holds at each stage of the derivation, we show that

(†) the property holds for the initial tableau

and

(‡) if the property holds for the tableau before application of a deduction rule, it also holds afterwards

That (†) is true is exactly the content of the *initial-tableau* proposition, that if the closed terms $\bar{t}[\bar{a}]$ satisfy the initial tableau (in the theory), then the program $\bar{f}(\bar{x}) = \bar{t}[\bar{x}]$ satisfies the specification.

To show (‡), we assume that

(*) the property holds for the tableau T before application of a deduction rule,

and show that then the property holds for the tableau T' after application of the rule. For this purpose, we suppose that

(**) the closed terms $\bar{t}[\bar{a}]$ satisfy the tableau T'

and show that then the program $\overline{f}(\overline{x}) = \overline{t}[\overline{x}]$ satisfies the specification.

Because the deduction rules maintain satisfying terms, we know, by our supposition (**), that $\overline{t}[\overline{a}]$ also satisfy the tableau $\mathcal{T}$. But then, by our assumption (*), the program $\overline{f}(\overline{x}) = \overline{t}[\overline{x}]$ satisfies our specification, as we wanted to show. ∎

THE FINAL TABLEAU

In a later section, we shall adapt each of our rules to apply to extended tableaux, in such a way that an extended justification condition will hold. The rules add new rows to the tableau, but the set of satisfying terms is the same at each stage. The deductive process continues until we obtain the final assertion

false		$\overline{t}'[\overline{a}]$

or the final goal

	true	$\overline{t}'[\overline{a}]$

where we require that the terms $\overline{t}'[\overline{a}]$ be primitive. These terms are not necessarily closed. Let $\overline{t}[\overline{a}]$ be obtained from $\overline{t}'[\overline{a}]$ by replacing all the variables with primitive constants, it does not matter which. At this point, we extract the ***final program***

$$\overline{f}(\overline{x}) = \overline{t}[\overline{x}].$$

Remark

If the terms $\overline{t}'[\overline{a}]$ are not primitive, we must continue the derivation until a final row goal *true* assertion *false* with primitive output entries is obtained. In fact, even if the $\overline{t}'[\overline{a}]$ are primitive, we may continue the derivation, perhaps to obtain a final row with different output entries, and hence a different final program. ∎

That the final program is indeed satisfactory is established as follows:

Proposition (final tableau)

The final program $\overline{f}(\overline{x}) = \overline{t}[\overline{x}]$ satisfies the specification $Q[\overline{x}; \overline{z}]$ in the theory. ∎

Proof

By the *intermediate-tableau* proposition, it suffices to show that the closed terms $\bar{t}[\bar{a}]$ satisfy the tableau in the theory. Let I be any model for the theory; it suffices to show that $\bar{t}[\bar{a}]$ suit the tableau under I. We actually show that $\bar{t}[\bar{a}]$ suit the final row.

We have taken $\bar{t}[\bar{a}]$ to be closed instances of the output entries $\bar{t}'[\bar{a}]$ of the final row. That is, $\bar{t}[\bar{a}]$ are $(\bar{t}'[\bar{a}]) \leftarrow \lambda$, for some substitution λ. Let us take our suiting substitution to be λ. To show the *truth* condition, we must show that the sentence $false \leftarrow \lambda$, that is, *false*, is closed and false under I [or the sentence $true \leftarrow \lambda'$, that is *true*, is closed and true under I]; but this is clearly the case.

To show the *output* condition, we must show that the instances $(\bar{t}'[\bar{a}]) \leftarrow \lambda$ are closed and have the same values, respectively, as $\bar{t}[\bar{a}]$ under I; but in fact these terms are respectively identical. ∎

Even before we describe the extension of the deduction rules, we illustrate the derivation process with a simple example.

Example (sex)

In the *family* theory, suppose that all people are either male or female, that is,

$$(\forall\ person\ u)[sex\,(u,\ male)\ \ or\ \ sex\,(u,\ female)] \qquad (sex)$$

Suppose we would like to construct a program $s(x) = t[x]$ to find the sex of a given person, that is, to meet the specification

$$Q[x;\ z]:\quad sex\,(x,\ z),$$

where the input sort is *person*.

From the specification, we form the initial tableau

assertions	goals	$s(a)$
	$G1.$ *if* $person\,(a)$ *then* $sex\,(a,\ z)$	z

We assume that the constants *male* and *female* and the predicate symbol *sex* are all primitive. We include the *sex* axiom as an assertion:

if person (u) *then sex* $(u,$ *male*$)$ *or sex* $(u,$ *female*$)$		

By the *initial-tableau* proposition, we know that, if any closed term $t[a]$ satisfies the above tableau in the *family* theory, the program $s(x) = t[x]$ must meet the specification.

By applying some extended deduction rules, we shall be able to obtain the goal row

	$G2$. *not sex* $(a,$ *female*$)$	*male*

By the *intermediate-tableau* proposition, we know that if any closed term $t[a]$ satisfies the new tableau in the *family* theory, the program $s(x) = t[x]$ again must meet the specification.

By applying another extended deduction rule, we shall be able to obtain the final goal row

	$G3$. *true*	*if sex* $(a,$ *female*$)$ *then female* *else male*

The conditional term (*if sex* $(a,$ *female*$)$ *then female else male*) is primitive and contains no variables. Therefore, we may stop the derivation process and derive the program

$$s(x) = \begin{cases} \textit{if sex}\,(x, \textit{female}) \\ \textit{then female} \\ \textit{else male} \end{cases}$$

By the *final-tableau* proposition, we know that this program must meet the specification. ∎

14.5 RECURSIVE PROGRAMS

Some special treatment is necessary to derive recursive programs $\overline{f}(\overline{x}) = \overline{t}[\overline{x}]$, in which the function symbols $\overline{f}$ may occur in the terms $\overline{t}[\overline{x}]$. Such programs are obtained by using the well-founded induction principle in the derivation.

We derive a program to satisfy the specification $Q[\overline{x}; \overline{z}]$ by proving a theorem $(\forall\ \overline{obj}\ \overline{x})(\exists\ \overline{z})Q[\overline{x}; \overline{z}]$. If we remove the outer quantifiers of this theorem in forming the initial tableau, we cannot complete the proof by induction on any of the variables $\overline{x}$. If instead we leave the quantifiers in place and invoke the induction principle before forming the initial tableau, we shall be able to obtain a recursive program.

The induction will be with respect to a well-founded relation over m-tuples of sort $\overline{obj}$. (See Section ??).

Definition (initial tableau, recursive)

In any theory, for a given specification $Q[\overline{x}; \overline{z}]$ with input sorts $\overline{obj}$ and for a given relation $\prec$ well-founded over m-tuples of sort $\overline{obj}$, an *initial tableau* for deriving a recursive program is as follows:

	goals	$\overline{f}(\overline{a})$
	if $\overline{obj}(\overline{a})$ *then if* $(\forall\ \overline{obj}\ \overline{u})$ [*if* $\langle\overline{u}\rangle \prec \langle\overline{a}\rangle$ *then* $Q[\overline{u}; \overline{f}(\overline{u})]$] *then* $Q[\overline{a}; \overline{z}]$	$\overline{z}$

Here $\overline{a} = a_1, \ldots, a_m$ are new constants. The function symbols $\overline{f}$ are included in the primitive list. Any valid sentences of the theory may be included in the tableau as assertions, without output entries. ∎

Example (flattree2)

In a combined theory of trees and strings, the specification for the program *flattree2*, which flattens a tree and concatenates it with a list, is

$$Q^2[x_1, x_2; z]: \ z = \mathit{flattree}(x_1) * x_2,$$

with input sorts *tree* and *string*. To construct a recursive program from this specification, with a well-founded relation $\prec$ over 2-tuples of trees and strings, we form the initial tableau

	goals	
	if $tree(a_1)$ *and* $string(a_2)$ *then if* $(\forall\ tree\ u_1)(\forall\ string\ u_2)\left[\begin{array}{l}\textit{if}\ \langle u_1, u_2\rangle \prec \langle a_1, a_2\rangle \\ \textit{then}\ \textit{flattree2}(u_1, u_2) \\ \qquad = \textit{flattree}(u_1) * u_2\end{array}\right]$ *then* $z = \textit{flattree}(a_1) * a_2$	z

Properties of the combined theory of trees and strings, may be included in the initial tableau as additional assertions. ⌟

The initial tableau for recursive programs may be shown to have the desired property.

Proposition (initial tableau for recursive programs)

In any theory,

> if the closed terms $\bar{t}[\bar{a}]$ satisfy the (recursive) initial tableau
> for a specification $Q[\bar{x}; \bar{z}]$ with input sorts $\overline{obj}$,
> and if $\prec$ is well-founded over m-tuples of sort $\overline{obj}$,
> then the program $\bar{f}(\bar{x}) = \bar{t}[\bar{x}]$ satisfies the specification. ⌟

Proof

To show that the program $\bar{f}(\bar{x}) = \bar{t}[\bar{x}]$ satisfies the specification, we establish the validity of the *correctness* condition

$$\begin{array}{l}\textit{if}\ (\forall\ \overline{obj}\ \bar{x})\left[\bar{f}(\bar{x}) = \bar{t}[\bar{x}]\right] \\ \textit{then}\ (\forall\ \overline{obj}\ \bar{x})Q\left[\bar{x}; \bar{f}(\bar{x})\right]\end{array}$$

in the theory.

Under a given model for the theory, we assume that

(*) $\qquad (\forall\ \overline{obj}\ \bar{x})\left[\bar{f}(\bar{x}) = \bar{t}[\bar{x}]\right].$

is true and show that then

$$(\forall\ \overline{obj}\ \bar{x})Q\left[\bar{x}; \bar{f}(\bar{x})\right]$$

is also true.

By the well-founded induction principle over m-tuples of sort $\overline{obj}$, it is enough to show the inductive step

$$(\forall\ \overline{obj}\ \overline{x}) \left[\begin{array}{l} \mathit{if}\ \ (\forall\ \overline{obj}\ \overline{u}) \left[\begin{array}{l} \mathit{if}\ \ \langle \overline{u} \rangle \prec \langle \overline{x} \rangle \\ \mathit{then}\ \ Q\left[\overline{u}; \overline{f}(\overline{u})\right] \end{array} \right] \\ \mathit{then}\ \ Q\left[\overline{x}; \overline{f}(\overline{x})\right] \end{array} \right].$$

It suffices, by our assumption (*), to show that

$$(\forall\ \overline{obj}\ \overline{x}) \left[\begin{array}{l} \mathit{if}\ \ (\forall\ \overline{obj}\ \overline{u}) \left[\begin{array}{l} \mathit{if}\ \ \langle \overline{u} \rangle \prec \langle \overline{x} \rangle \\ \mathit{then}\ \ Q\left[\overline{u}; \overline{f}(\overline{u})\right] \end{array} \right] \\ \mathit{then}\ \ Q\left[\overline{x}; \overline{t}[\overline{x}]\right] \end{array} \right]$$

or, expanding the outermost relativized quantifier,

$$(\forall\ \overline{x}) \left[\begin{array}{l} \mathit{if}\ \ \overline{obj}(\overline{x}) \\ \mathit{then}\ \ \mathit{if}\ \ (\forall\ \overline{obj}\ \overline{u}) \left[\begin{array}{l} \mathit{if}\ \ \langle \overline{u} \rangle \prec \langle \overline{x} \rangle \\ \mathit{then}\ \ Q\left[\overline{u}; \overline{f}(\overline{u})\right] \end{array} \right] \\ \qquad \mathit{then}\ \ Q\left[\overline{x}; \overline{t}[\overline{x}]\right] \end{array} \right].$$

is true.

By the *output entry* property, to show that the above sentence is actually valid, it suffices to find closed terms $\overline{t}[\overline{a}]$ that satisfy the tableau

	goals	$\overline{f}(\overline{a})$
	$\mathit{if}\ \ \overline{obj}(\overline{a})$ $\mathit{then}\ \ \mathit{if}\ \ (\forall\ \overline{obj}\ \overline{u}) \left[\begin{array}{l} \mathit{if}\ \ \langle \overline{u} \rangle \prec \langle \overline{a} \rangle \\ \mathit{then}\ \ Q\left[\overline{u}; \overline{f}(\overline{u})\right] \end{array} \right]$ $\qquad \mathit{then}\ \ Q[\overline{a}; \overline{z}]$	$\overline{z}$

The *valid-assertion* property allows us to add as assertions to this tableau any valid sentences of the theory, without changing the satisfying terms. Thus, it suffices that the closed terms $\overline{t}[\overline{a}]$ satisfy the (recursive) initial tableau, as we have assumed. ∎

Remark (one input case)

⟪ MB: too trivial to be mentioned; OUT ⟫

In the special case in which there is only one input object (i.e., $m = 1$), we have no need to use well-founded induction over tuples of sort $\overline{obj}$; we may use well-founded induction over individuals of sort obj_1, that is, obj, instead. (Recall that in this case we drop the subscripts from the input symbols.) For the specification $Q[x;\overline{z}]$, we take our (recursive) initial tableau to be

	goals	$\overline{f}(a)$
	if $obj(a)$ *then if* $(\forall\ obj\ u)$ [*if* $u \prec a$ *then* $Q[u;\overline{f}(u)]$] *then* $Q[a;\overline{z}]$	$\overline{z}$

where $\prec$ is well-founded over obj.

By the same argument as for the general case, we may show that if the closed terms $\overline{t}[a]$ satisfy this tableau (with optional valid sentences of the theory added assertions), then the program $\overline{f}(x) = \overline{t}[x]$ satisfies the specification. ∎

14.6 THE DEDUCTION RULES

We are now ready to adapt the deduction rules of our system to apply to extended tableaux. Each rule will introduce new output entries as well as assertions or goals and will maintain satisfying terms as well as validity.

JUSTIFICATION PROPOSITION

Each deduction rule requires that certain rows (the "required rows") already be present in the tableau and generates certain new rows (the "generated rows") to be introduced into the tableau. For each deduction rule we require an *extended justification condition* which consists of the original *justification* condition, which guarantees that the rule maintains validity, plus a new condition, which guarantees that the rule maintains satisfying terms.

Proposition (justification)

A deduction rule maintains the terms satisfying a tableau to which it is applied if the following *justification condition for satisfying terms* holds:

For any model I for the theory and for any closed terms $\bar{t}$,

> if $\bar{t}$ suit any of the generated rows under I
> then $\bar{t}$ suit at least one of the required rows under I. ∎

Proof

Assume that the *justification condition for satisfying terms* holds; we would like to show that the deduction rule maintains satisfying terms.

Consider arbitrary closed terms $\bar{t}$. We show that, in the theory, the terms $\bar{t}$ satisfy the given tableau $\mathcal{T}$ if and only if they satisfy the new tableau $\mathcal{T}'$.

In one direction, the proof does not require the justification condition at all. Suppose that the terms $\bar{t}$ satisfy the original tableau $\mathcal{T}$ in the theory. Let I be any model for the theory, then $\bar{t}$ suit some row of $\mathcal{T}$ under I. But because deduction rules do not delete rows, every row of $\mathcal{T}$ is also a row of $\mathcal{T}'$. So $\bar{t}$ suit some row of $\mathcal{T}'$, that is, $\bar{t}$ suit $\mathcal{T}'$ under I. Hence $\bar{t}$ satisfy the new tableau $\mathcal{T}'$, as we wanted to show.

For the other direction, suppose that the terms $\bar{t}$ satisfy the new tableau $\mathcal{T}'$. We would like to show that then $\bar{t}$ satisfy the original tableau $\mathcal{T}$. Let I be any model for the theory; then it suffices to show that $\bar{t}$ suit $\mathcal{T}$ under I.

Because the terms $\bar{t}$ satisfy $\mathcal{T}'$ in the theory, $\bar{t}$ suit some row of $\mathcal{T}'$ under I. If this row was already a row of $\mathcal{T}$, then $\bar{t}$ suit $\mathcal{T}$ under I, as we wanted to show. Otherwise, the row must have been generated by the deduction rule. In this case (by the *justification* condition for satisfying terms), $\bar{t}$ must suit at least one of the required rows, which must appear in $\mathcal{T}$. Hence $\bar{t}$ suit $\mathcal{T}$ under I, as we wanted to show. ∎

SIMPLIFICATION

Any sentence or term introduced into a tableau is automatically subjected to a simplification process, in which certain subsentences are replaced by equivalent but simpler sentences, and certain subterms are replaced by equal but simpler terms. For extended tableaux, simplification is applied to the output entries as well as the assertions and goals. Simplification is not regarded as a separate rule; we apply it automatically whenever we add a new row to a tableau.

Example

Suppose that the following row is to be added to a tableau:

assertions	goals	$f(a)$
	$p(x)$ *and* $p(x)$	*if* $p(x)$ *then* a *else* a

The goal of this row would automatically be simplified to $p(x)$, by application of the simplification

$$(\mathcal{F} \textit{ and } \mathcal{F}) \Rightarrow \mathcal{F}.$$

Also, the output entry would be simplified to a, by application of the simplification
⟪ do we simplify terms ⟫

$$(\textit{if } \mathcal{F} \textit{ then } s \textit{ else } s) \Rightarrow s.$$

The entire row would thus be simplified to

	$p(x)$	a

Simplification is easily justified because we always replace a subsentence by an equivalent sentence or a subterm by an equal term. In particular, the satisfying terms are maintained.

SPLITTING RULES

The splitting rules break rows down into their logical components. The extended rules are very similar to the basic splitting rules. The output entries of the generated rows are the same as those of the required rows. We present all three splitting rules in tableau notation.

- *And-split rule*

assertions	goals	$\overline{f}(\overline{a})$
A_1 *and* A_2		$\overline{s}$
A_1		$\overline{s}$
A_2		$\overline{s}$

- *Or-split rule*

assertions	goals	$\overline{f}(\overline{a})$
	G_1 *or* G_2	$\overline{s}$
	G_1	$\overline{s}$
	G_2	$\overline{s}$

- *If-split rule*

assertions	goals	$\overline{f}(\overline{a})$
	if A *then* G	$\overline{s}$
A		$\overline{s}$
	G	$\overline{s}$

In each rule, there are n output columns, containing the output entries $\overline{s} = s_1, s_2, \ldots, s_n$.

Remark

In deriving a nonrecursive program from a specification $Q[\overline{x}, \overline{z}]$, we formed the initial tableau

assertions	goals	$\overline{f}(\overline{a})$
	if $\overline{obj}(\overline{a})$ *then* $Q[\overline{a}, \overline{z}]$	$\overline{z}$

By application of the *if-split* rule, we may decompose this row into an assertion and a goal

$\overline{obj}(\overline{a})$		$\overline{z}$
	$Q[\overline{a}; \overline{z}]$	$\overline{z}$

Because the output entries $\overline{z}$ do not occur free in the assertion, they may be dropped; of course, the output entries for the goal must remain.

We shall automatically apply the *if-split* rule to the initial tableau. In fact, we shall henceforth regard the initial tableau to be the resulting assertion and goal, that is,

assertions	goals	$\overline{f}(\overline{a})$
$\overline{obj}(\overline{a})$		
	$Q[\overline{a}; \overline{z}]$	$\overline{z}$

Similarly, in deriving a recursive program from a specification $Q[\overline{x}; \overline{z}]$, we took the initial tableau to be

	goals	$f(\overline{x})$
	if $obj(\overline{a})$ *then if* $(\forall\ \overline{obj}\ \overline{u})$ $\begin{bmatrix} \textit{if}\ \ (\overline{u}) \prec (\overline{a}) \\ \textit{then}\ \ Q[\overline{u}; \overline{f}(\overline{u})] \end{bmatrix}$ *then* $Q[\overline{a}; \overline{z}]$	$\overline{z}$

By application of the *if-split* rule, we may decompose this row to obtain an assertion and a goal

assertions	goals	$\overline{f}(\overline{a})$
$\overline{obj}(\overline{a})$		$\overline{z}$
	if $(\forall\ \overline{obj}\ \overline{u})$ $\begin{bmatrix} \textit{if}\ \ (\overline{u}) \prec (\overline{a}) \\ \textit{then}\ \ Q[\overline{u}; \overline{f}(\overline{u})] \end{bmatrix}$ *then* $Q[\overline{a}; \overline{z}]$	$\overline{z}$

By a second application of the *if-split* rule, we may decompose the goal further, to obtain

$(\forall\ \overline{obj}\ \overline{u})\begin{bmatrix} \textit{if}\ \ \langle \overline{u}\rangle \prec \langle \overline{a}\rangle \\ \textit{then}\ \ Q[\overline{u};\ \overline{f}(\overline{u})] \end{bmatrix}$		$\overline{z}$
	$Q[\overline{a};\ \overline{z}]$	$\overline{z}$

The new assertion corresponds to the induction hypothesis, and the new goal to the derived conclusion, for the inductive step of a proof. Again, because the output entries $\overline{z}$ do not occur free in the assertions, the output entries for these rows may be dropped.

We shall automatically apply the *if-split* rule to the initial tableau for a recursive program; in fact, we shall henceforth regard the following three rows as the initial tableau:

assertions	goals	$\overline{f}(\overline{a})$
$\overline{obj}(\overline{a})$		
$(\forall\ \overline{obj}\ \overline{u})\begin{bmatrix} \textit{if}\ \ \langle \overline{u}\rangle \prec \langle \overline{a}\rangle \\ \textit{then}\ \ Q[\overline{u};\ \overline{f}(\overline{u})] \end{bmatrix}$		
	$Q[\overline{a};\ \overline{z}]$	$\overline{z}$

Example (initial tableaux after splitting) ⟪ necessary? MB ⟫

For the *flattree*1 program, which flattens a tree, the specification is

$$Q[x;\ z]:\ z = \textit{flattree}\,(x),$$

with input sort *tree*. To construct a (nonrecursive) program to meet this specification, we form the initial tableau

assertions	goals	*flattree*1(a)
tree(a)		
	$z = \textit{flattree}\,(x)$	z

As usual, valid sentences of the theory may be included as additional assertions. ⟪ out?? MB ⟫

For the *flattree2* program, which flattens a tree and concatenates the result with another string, the specification is

$$\mathcal{Q}[x_1,\ x_2;\ z]:\ z = \mathit{flattree}\,(x_1) * x_2,$$

with input sorts *tree* and *string*. To construct a recursive program to meet this specification, we form the initial tableau

assertions	goals	
$\mathit{tree}(a_1)$ *and* $\mathit{string}(a_2)$		
$(\forall\ \mathit{tree}\ u_1)$ $(\forall\ \mathit{string}\ u_2)$ [*if* $(u_1, u_2) \prec (a_1, a_2)$ *then* $\mathit{flattree2}(u_1, u_2) = \mathit{flattree}\,(u_1) * u_2$]		
	$z = \mathit{flattree}\,(a_1) * a_2$	z

$\mathit{flattree2}(a_1, a_2)$

Again, valid sentences of the theory may be included as additional assertions. ⟪ out?? MB ⟫ ⌟

We shall justify only the *if-split* rule.

Justification (if-split rule)

We would like to establish that the *justification* conditions hold for the *if-split* rule. The *justification* condition for validity was established when we introduced the rule for basic tableaux. We need only show the *justification* condition for satisfying terms.

Let I be a model for the theory in question and let $\bar{t}$ be closed terms that suit one of the generated rows under I. We would like to show that $\bar{t}$ suit the required goal (*if* A *then* $\mathcal{G}$) under I.

If the terms $\bar{t}$ suit the generated assertion A under I, then for some suiting substitution λ, we have the *truth* condition,

$A \leftarrow \lambda$ is closed and false under I,

and the *output* condition,

$\bar{s} \leftarrow \lambda$ are closed and have the same values, respectively, as $\bar{t}$ under I.

Suppose $u_1, \ldots, u_k$ are the free variables of $\mathcal{G} \leftarrow \lambda$ and let $\overline{\lambda} = \{u_1 \leftarrow a, \ldots, u_k \leftarrow a\}$, where a is any constant. We claim that, as we wanted to show, the terms $\bar{t}$ suit

the goal (*if* A *then* $\mathcal{G}$), with suiting substitution $\lambda \sqcup \overline{\lambda}$. The *truth* condition holds, that is,

$$(\textit{if } A \textit{ then } \mathcal{G}) \blacktriangleleft (\lambda \sqcup \overline{\lambda}) = \textit{if } (A \blacktriangleleft \lambda) \blacktriangleleft \overline{\lambda} \textit{ then } (\mathcal{G} \blacktriangleleft \lambda) \blacktriangleleft \overline{\lambda}$$

$$= \textit{if } A \blacktriangleleft \lambda \textit{ then } (\mathcal{G} \blacktriangleleft \lambda) \blacktriangleleft \overline{\lambda}$$

(because $A \blacktriangleleft \lambda$ is closed)

is closed and true under I (since $A \blacktriangleleft \lambda$ is false under I),

and the *output* condition holds, that is,

$$\overline{s} \blacktriangleleft (\lambda \sqcup \overline{\lambda}) = (\overline{s} \blacktriangleleft \lambda) \blacktriangleleft \overline{\lambda}$$

$$= \overline{s} \blacktriangleleft \lambda$$

(because $\overline{s} \blacktriangleleft \lambda$ are closed)

are closed and have the same values, respectively, as $\overline{t}$ under I.

Similarly, if the terms $\overline{t}$ suit the generated goal $\mathcal{G}$ under I, with suiting substitution λ, we can construct a substitution $\overline{\lambda}$ such that $\overline{t}$ suit the required goal (*if* A *then* $\mathcal{G}$), with suiting substitution $\lambda \sqcup \overline{\lambda}$. ∎

THE RESOLUTION RULE

The resolution rule allows us to perform a case analysis on the truth of a subsentence. In extending the rule, the output entries for the generated row are, in general, conditional terms.

Let us write the rule in tableau notation.

Rule (AA-resolution)

assertions	goals	$\overline{f}(\overline{a})$
A_1		$\overline{s}$
A_2		$\overline{t}$
$(A_1 \blacktriangleleft \theta) \blacktriangleleft \{P \blacktriangleleft \theta \leftarrow \textit{false}\}$ *or* $(A_2 \blacktriangleleft \theta) \blacktriangleleft \{P \blacktriangleleft \theta \leftarrow \textit{true}\}$		*if* $P \blacktriangleleft \theta$ *then* $\overline{t} \blacktriangleleft \theta$ *else* $\overline{s} \blacktriangleleft \theta$

∎

⟪ informal notation? ⟫

The notation and requirements for the rule and the new assertion introduced are the same as for the rule without output entries.

The new output entries are conditional terms, each of whose *if*-clauses is the unified subsentence $P \lhd \theta$, and whose *then*- and *else*-clauses are the corresponding instances $\overline{t} \lhd \theta$ and $\overline{s} \lhd \theta$, respectively, of the output entries $\overline{t}$ and $\overline{s}$ for the required rows. In other words,

$$\textit{if } P \lhd \theta \textit{ then } \overline{t} \lhd \theta \textit{ else } \overline{s} \lhd \theta$$

is an abbreviation for the n terms

$$\begin{array}{lll} \textit{if } P \lhd \theta & \textit{if } P \lhd \theta & \textit{if } P \lhd \theta \\ \textit{then } t_1 \lhd \theta & \textit{then } t_2 \lhd \theta & \textit{then } t_n \lhd \theta \\ \textit{else } s_1 \lhd \theta & \textit{else } s_2 \lhd \theta & \textit{else } s_n \lhd \theta. \end{array} \quad \cdots$$

As usual, by duality, the rule can be applied to two goals or to an assertion and a goal; the output entries are the same as for the AA version of the rule. The polarity strategy is as before. In the case in which one (or both) of the two required rows has no output entries, the row is treated as if it has output entries $\overline{u} = u_1, u_2, \ldots, u_n$, where none of the variables u_i occur free in the row.

We first look at an example that does not require unification.

Example (no unification)

Suppose we have the two rows

assertions	goals	$rh(a_1, a_2)$	$nrh(a_1, a_2)$
	$not\ \boxed{red(a_2)}^-$	a_1	a_2
	$\boxed{red(a_2)}^+$	a_2	a_3

The boxed subsentences of the two goals are identical, and hence unifiable with most-general unifier { }. Therefore we may apply the resolution rule to obtain

	not false *and* *true*	*if* $red(a_2)$ *then* a_2 *else* a_1	*if* $red(a_2)$ *then* a_3 *else* a_2

which is automatically simplified to

	true	*if* $red(a_2)$ *then* a_2 *else* a_1	*if* $red(a_2)$ *then* a_3 *else* a_2

The output entries are conditional terms each of whose *if*-clause is the unified subsentence $red(a_2)$. The *then*-terms and *else*-terms are the output entries of the given rows (in reverse order). ∎

Now let us see an example in which a unifier is necessary to create common subsentences in the goals.

Example (with unification)

⟪ ms. unclear here - eed??? ⟫ In a derivation for the *sex* program $s(x)$, we obtain the two rows

assertions	goals	$s(a)$
	not $\boxed{sex(a, female)}^-$	*male*
	$\boxed{sex(a, z)}^+$	z

The boxed subsentences of the two goals are unifiable, with most-general unifier $\{z \leftarrow female\}$. Therefore, we may apply the resolution rule to obtain

	not false *and* *true*	*if* $sex(a, female)$ *then* *female* *else* *male*

which is automatically simplified to

	true	*if* $sex(a, female)$ *then* *female* *else* *male*

Note that the most-general unifier has been applied to the output entries as well as the goals. ∎

Now let us see an example in which one of the given rows has no output entry.

Example (one output entry)

Suppose we have the rows

assertions	goals	$s(a)$
	$sex\,(a, z)$	z
$sex\,(u, male)$ *or* $sex\,(u, female)$		

Note that the second initial assertion has no output entry. We may therefore treat it as if it had the new variable u_1 as its output entry.

$\boxed{sex\,(u, male)}^{-}$ *or* $sex\,(u, female)$		u_1

The boxed subsentence of this assertion is unifiable with the goal

	$\boxed{sex\,(a, z)}^{+}$	z

A most-general unifier is $\{u \leftarrow a, z \leftarrow male\}$. Therefore, we may apply the resolution rule to obtain

	not (*false or* $sex\,(a, female)$) *and* *true*	*if* $sex\,(a, male)$ *then male* *else* u_1

which is automatically simplified to

	not $sex\,(a, female)$	*if* $sex\,(a, male)$ *then male* *else* u_1

Remark

Suppose we apply the resolution rule to two rows whose ith output entries s_i and t_i become identical after the unifying substitution θ is applied, that is, the terms $s_i \blacktriangleleft \theta$ and $t_i \blacktriangleleft \theta$ are identical. Then the ith output entry of the generated row, (*if* $P \blacktriangleleft \theta$ *then* $t_i \blacktriangleleft \theta$ *else* $s_i \blacktriangleleft \theta$), is automatically simplified to $s_i \blacktriangleleft \theta$, by

application of the simplification

$$(\textit{if } \mathcal{F} \textit{ then } s \textit{ else } s) \Rightarrow s.$$

Suppose in applying the resolution rule we generate output entries of form

$$(\textit{if } \mathcal{F} \textit{ then } u \textit{ else } s) \quad \text{or} \quad (\textit{if } \mathcal{F} \textit{ then } s \textit{ else } u),$$

where u is a variable that does not occur free elsewhere in the row. This occurs when we apply the rule to a row without output entries. Then we shall automatically replace those output entries with s.

To justify this, observe that, by the *instantiation* property, we may apply the substitution $\lambda\{u \leftarrow s\}$ to the generated row. This transforms the output entries into $(\textit{if } \mathcal{F} \textit{ then } s \textit{ else } s)$ and has no effect on the remainder of the row. We may then simplify the output entries, obtaining s.

In particular, in applying the resolution rule to rows that do not both have output entries, we do not actually introduce conditional terms into the output columns. If only one of the rows has output entries $\bar{t}$, the generated row has output entries $\bar{t} \cdot \theta$, where θ is the unifying substitution. ∎

Example

In the previous example, we have obtained the row

assertions	goals	$s(a)$
	not sex $(a,$ *female*$)$	*if sex* $(a,$ *male*$)$ *then male* *else* u_1

Note that the variable u_1 in the output entry does not occur elsewhere in the row. In accordance with the preceding remark, we can replace the output entry of this row with *male*, to obtain

	3. *not sex* $(a,$ *female*$)$	*male*

∎

Remark (no output case)

Suppose we are applying the resolution rule to two rows, both of which have no output entries. We treat each row as if it had as its output entries the new

variables $\overline{u}$ and $\overline{v}$. The rule will then produce a row whose output entries are of form (*if* $P \blacktriangleleft \theta$ *then* $\overline{v}$ *else* $\overline{u}$). These output entries are automatically replaced by the output entries $\overline{u}$, in accordance with our previous remark. The variables $\overline{u}$, being new, do not occur in the newly generated row. For this reason, the new row will be treated as if it had no output entries.

In short, if the resolution rule is applied to two rows without output entries, the resulting row has no output entries either. ∎

Let us now justify our adaptation of the resolution rule to extended tableaux.

Justification (extended resolution rule)

We consider only the AA-form of the rule and show that the *justification* conditions hold. The *justification* condition for validity was established when we introduced the basic resolution rule.

To show the *justification* condition for satisfying terms, let I be a model for the theory and let $\overline{r}$ be closed terms that suit the generated assertion under I. We would like to show that then $\overline{r}$ suit one of the required assertions under I.

Because the terms $\overline{r}$ suit the generated assertion under I, we have, for some suiting substitution λ, the *truth* condition,

$$\Big[(A_1 \blacktriangleleft \theta) \blacktriangleleft \{P \blacktriangleleft \theta \leftarrow \mathit{false}\} \ \mathit{or}\ (A_2 \blacktriangleleft \theta) \blacktriangleleft \{P \blacktriangleleft \theta \leftarrow \mathit{true}\}\Big] \blacktriangleleft \lambda$$
is closed and false under I,

and the *output* condition,

the terms (*if* $P \blacktriangleleft \theta$ *then* $\overline{t} \blacktriangleleft \theta$ *else* $\overline{s} \blacktriangleleft \theta$) $\blacktriangleleft \lambda$ are closed and have the same values, respectively, as $\overline{r}$ under I.

Consequently (by properties of substitutions), we have

(†) $[A_1 \blacktriangleleft (\theta \square \lambda)] \blacktriangleleft \{P \blacktriangleleft (\theta \square \lambda) \leftarrow \mathit{false}\}$
is closed and false under I,

$[A_2 \blacktriangleleft (\theta \square \lambda)] \blacktriangleleft \{P \blacktriangleleft (\theta \square \lambda) \leftarrow \mathit{true}\}$
is closed and false under I,

⟪ clarify these last two steps earlier ⟫

and

(‡) the terms *if* $P \blacktriangleleft (\theta \square \lambda)$ *then* $\overline{t} \blacktriangleleft (\theta \square \lambda)$ *else* $\overline{s} \blacktriangleleft (\theta \square \lambda)$
are closed and have the same values, respectively, as $\overline{r}$ under I.

We distinguish between two cases.

Case: $P \lhd (\theta \square \lambda)$ is false under I.

We claim that then the terms $\overline{r}$ suit the required assertion A_1 with suiting substitution $\theta \square \lambda$. To show this, we must show the appropriate *truth* and *output* conditions.

In this case, $P \lhd (\theta \square \lambda)$ and *false* have the same truth-value under I. Because $\left[A_1 \lhd (\theta \square \lambda)\right] \lhd \{P \lhd (\theta \square \lambda) \leftarrow \mathit{false}\}$ is false under I (by (†)), it follows (by the *value* property) that the closed ⟪ why? ⟫ sentence $A_1 \lhd (\theta \square \lambda)$ is false under I, that is, the *truth* condition holds.

⟪ use *value* property in previous proofs ⟫

Also, because in this case $P \lhd (\theta \square \lambda)$ is false under I, it follows (from (‡)) that the closed terms $\overline{s} \lhd (\theta \square \lambda)$ have the same values, respectively, as $\overline{r}$ under I, that is, the *output* condition also holds.

Case: $P \lhd (\theta \square \lambda)$ is true under I.

This case is symmetric to the previous case. We show that the terms $\overline{r}$ suit the required assertion A_2 with suiting substitution $\theta \square \lambda$. ∎

THE EQUIVALENCE RULE

The equivalence rule allows us to replace a subsentence of the tableau with an equivalent sentence. For extended tableaux, we may replace subsentences of the output entries as well as the assertions and goals. The output entries for the generated row are, in general, conditional terms. Let us write the rule in tableau form.

Rule (AA-equivalence, left to right)

assertions		$\overline{f}(\overline{a})$
A_1		$\overline{s}$
A_2		$\overline{t}$
$(A_1 \lhd \theta) \lhd \{(P \equiv Q) \lhd \theta \leftarrow \mathit{false}\}$ *or* $(A_2 \lhd \theta) \lhd \{P \lhd \theta \leftarrow Q \lhd \theta\}$		*if* $(P \equiv Q) \lhd \theta$ *then* $(\overline{t} \lhd \theta) \lhd \{P \lhd \theta \leftarrow Q \lhd \theta\}$ *else* $\overline{s} \lhd \theta$

⟪ ◂ and ◂ not as in Vol. 1, p. 43 ⟫

Note that the substitution θ may unify occurrences of subsentences in the output entries $\bar{t}$ as well as in the assertion A_2 and these occurrences may be unified by the rule. (An output entry may have subsentences if it contains a conditional term.) A right-to-left version of the rule allows us to replace occurrences of $Q \blacktriangleleft \theta$ with $P \blacktriangleleft \theta$.

We have presented the rule as it applies to two assertions. As usual, by duality, we may also apply the rule to an assertion and a goal, or to two goals, and obtain conditional output entries in each case. As was the case for the resolution rule, if one of the two given rows has no output entries, the conditional is not introduced into the output entries for the generated row. If both given rows have no output entries, the generated rows have no output entries either. The polarity strategy applies as usual.

Example (equivalence rule)

Suppose our tableau contains the two rows

assertions	goals	$\overline{f}(\bar{a})$
$\left[\boxed{p(x, a)} \equiv q(x)\right]^{-}$		$g(x)$
$\boxed{p(b, y)}$		*if* $\boxed{p(z, y)}$ *then* z *else* y

The boxed subsentences of the two rows are unifiable, with a most-general unifier

$$\{x \leftarrow b,\ y \leftarrow a,\ z \leftarrow b\}.$$

We may apply the AA-*equivalence* rule to obtain

false *or* $q(b)$		*if* $p(b, a) \equiv q(b)$ *then if* $q(b)$ *then* b *else* a *else* $g(b)$

or, after *true-false* simplification,

$q(b)$		*if* $p(b, a) \equiv q(b)$ *then if* $q(b)$ *then* b *else* a *else* $g(b)$

Here we have replaced an occurrence of $p(b, a)$ with $q(b)$ in the output entry as well as in the assertion. It would also have been legal for us to make a replacement in the assertion but not the output entry, or in the output entry but not the assertion. The unifiers would have been different in each case. ∎

The justification for the extended *equivalence* rule is analogous to that for the extended *resolution* rule. ⟪ exercise! ⟫

Justification (extended *equivalence* rule)

We consider *only the* AA, left-to-right version of the rule and show that the ***justification*** conditions hold. The ***justification*** condition for validity was established when we introduced the basic equivalence rule.

To show the ***justification*** condition for satisfying terms, let I be a model for the theory and $\bar{r}$ be closed terms that suit the generated assertion under I. We would like to show that then $\bar{r}$ suit one of the required assertions under I.

Because the terms $\bar{r}$ suit the generated assertion under I, we know that, for some suiting substitution λ, we have the *truth* condition,

$$\begin{bmatrix} (A_1 \lhd \theta) \lhd \{(P \equiv Q) \lhd \theta \leftarrow \mathit{false}\} \\ \quad \mathit{or} \\ (A_2 \lhd \theta) \lhd \{P \lhd \theta \leftarrow Q \lhd \theta\} \end{bmatrix} \lhd \lambda$$

is closed and false under I,

and the ***output*** condition

the terms

$$\begin{bmatrix} \mathit{if}\ (P \equiv Q) \lhd \theta \\ \mathit{then}\ (\bar{t} \lhd \theta) \lhd \{P \lhd \theta \leftarrow Q \lhd \theta\} \\ \mathit{else}\ \bar{s} \lhd \theta \end{bmatrix} \lhd \lambda$$

are closed and have the same values, respectively, as $\bar{r}$ under I.

Consequently (by properties of substitutions ⟪ ?? ⟫), we have ⟪ small box? check subst. chapter ⟫

(*) $[A_1 \blacktriangleleft (\theta \square \lambda)] \blacktriangleleft \{(P \blacktriangleleft (\theta \square \lambda) \equiv Q \blacktriangleleft (\theta \square \lambda)) \leftarrow \mathit{false}\}$
is closed and false under I,

(†) $[A_2 \blacktriangleleft (\theta \square \lambda)] \blacktriangleleft \{P \blacktriangleleft (\theta \square \lambda) \leftarrow Q \blacktriangleleft (\theta \square \lambda)\}$
is closed and false under I,

and

(‡) the terms
if $P \blacktriangleleft (\theta \square \lambda) \equiv Q \blacktriangleleft (\theta \square \lambda)$
then $[\bar{t} \blacktriangleleft (\theta \square \lambda)] \blacktriangleleft \{P \blacktriangleleft (\theta \square \lambda) \leftarrow Q \blacktriangleleft (\theta \square \lambda)\}$
else $\bar{s} \blacktriangleleft (\theta \square \lambda)$
are closed and have the same values, respectively, as $\bar{r}$ under I.

We distinguish between two cases.

Case: $(P \blacktriangleleft (\theta \square \lambda) \equiv Q \blacktriangleleft (\theta \square \lambda))$ is false under I.

We claim that then the terms $\bar{r}$ suit the required assertion A_1 with suiting substitution $\theta \square \lambda$. To show this, we establish the appropriate *truth* and *output* conditions.

In this case, $(P \blacktriangleleft (\theta \square \lambda) \equiv Q \blacktriangleleft (\theta \square \lambda))$ and *false* have the same truth-value under I. Because $[A_1 \blacktriangleleft (\theta \square \lambda)] \blacktriangleleft \{(P \blacktriangleleft (\theta \square \lambda) \equiv Q \blacktriangleleft (\theta \square \lambda)) \leftarrow \mathit{false}\}$ is false under I (by (*)), it follows (by the *value* property) that the closed ⟪ why? ⟫ sentence $A_1 \blacktriangleleft (\theta \square \lambda)$ is false under I, that is, the *truth* condition holds.

Also, because in this case $(P \blacktriangleleft (\theta \square \lambda) \equiv Q \blacktriangleleft (\theta \square \lambda))$ is false under I, it follows (from (‡)) that the closed terms $\bar{s} \blacktriangleleft (\theta \square \lambda)$ have the same values, respectively, as $\bar{r}$ under I, that is, the *output* condition also holds.

Case: $(P \blacktriangleleft (\theta \square \lambda) \equiv Q \blacktriangleleft (\theta \square \lambda))$ is true under I.

We claim that the terms $\bar{r}$ suit the required assertion A_2 with suiting substitution $\theta \square \lambda$. To show this, we again establish the appropriate *truth* and *output* conditions.

In this case, $P \blacktriangleleft (\theta \square \lambda)$ and $Q \blacktriangleleft (\theta \square \lambda)$ have the same truth-value under I. Because

$$[A_2 \blacktriangleleft (\theta \square \lambda)] \blacktriangleleft \{P \blacktriangleleft (\theta \square \lambda) \leftarrow Q \blacktriangleleft (\theta \square \lambda)\}$$

is false under I (by (†)), it follows (by the *value* property) that the closed ⟪ why? ⟫ sentence $A_2 \blacktriangleleft (\theta \square \lambda)$ is false under I, that is, the *truth* condition holds.

Also, because in this case $(P \lhd (\theta \square \lambda) \equiv Q \lhd (\theta \square \lambda))$ is true under I, it follows (from (‡)) that the closed terms

$$[\bar{t} \lhd (\theta \square \lambda)] \lhd \{P \lhd (\theta \square \lambda) \leftarrow Q \lhd (\theta \square \lambda)\}$$

have the same values, respectively, as $\bar{r}$ under I. Because in this case $P \lhd (\theta \square \lambda)$ and $Q \lhd (\theta \square \lambda)$ have the same truth-value under I, it follows that the closed ⟪ why? ⟫ terms $\bar{t} \lhd (\theta \square \lambda)$ have the same values, respectively, as $\bar{r}$ under I, that is, the *output* condition also holds. ∎

THE EQUALITY RULE

The equality rule is analogous to the equivalence rule: it allows us to replace a subterm of the tableau with an equal term. For tableaux with output columns, we may replace subterms of the output entries as well as the assertions and goals. The output entries for the generated row are, in general, conditional terms.

The rule allows us to omit the *transitivity* and *symmetry* axioms for equality from our list of assertions; the *reflexivity* axiom $u = u$ is retained.

Let us write the rule in tableau form.

Rule (AA-equality, left to right)

assertions		$\overline{f}(\bar{a})$
A_1		$\bar{s}$
A_2		$\bar{t}$
$(A_1 \lhd \theta) \lhd \{(p = q) \lhd \theta \leftarrow \mathit{false}\}$ *or* $(A_2 \lhd \theta) \lhd \{p \lhd \theta \leftarrow q \lhd \theta\}$		*if* $(p = q) \lhd \theta$ *then* $(\bar{t} \lhd \theta) \lhd \{p \lhd \theta \leftarrow q \lhd \theta\}$ *else* $\bar{s} \lhd \theta$

∎

⟪ Here the symbols p and q stand for terms. ⟫ A right-to-left version of the rule allows us to replace occurrences of $q \lhd \theta$ with $p \lhd \theta$.

We have presented the rule as it applies to two assertions but, by duality, it may be applied as well to two goals or to an assertion and a goal. As usual, if only one of the rows has an output entry, the conditional is not introduced into

the generated output entry; if neither row has output entries, the generated row also has no output entry. The polarity strategy applies as before.

We omit the justification for the equality rule, since it is closely analogous to that for the equivalence rule.

Example (equality rule)

Suppose our tableau contains the two rows

assertions	goals	$\overline{f}(\overline{a})$
$\left[\boxed{0+x} = x\right]^{-}$		
	$\boxed{z+a} = b$	z

The boxed subsentences of the two rows are unifiable, with a most-general unifier

$\{x \leftarrow a, z \leftarrow 0\}$.

We may apply the AG-*equality* rule to obtain

	not false *and* $a = b$	0

which simplifies to

	$a = b$	0

Note that because the assertion had no output entry, we did not introduce a conditional into the generated row. ∎

THE SKOLEMIZATION RULES

The *universal* and *existential quantifier-elimination* propositions are invoked in forming the initial tableau, because we remove the quantifiers for the input and output variables before the tableau is formed. Nevertheless, there may be other

quantifiers in the specification that must be removed once the derivation is underway. For this purpose, we can apply the ***quantifier-elimination*** rules, which allow us to remove quantifiers of strict force from the assertions and goals. The output entries remain the same.

Rules (quantifier elimination)

assertions	goals	$\overline{f}(\overline{a})$
A		$\overline{s}$
A'		$\overline{s}$

Here A' is obtained from A by dropping a quantifier of strict force, either universal or existential, as in the basic $\forall$- and $\exists$-*elimination* rules. Precisely the same rules apply to goals. ∎

Remark

In forming the initial tableau for a recursive program, we introduced into the initial tableau the induction hypothesis

$(\forall\ \overline{obj}\ \overline{u})\begin{bmatrix} \textit{if}\ \ \langle\overline{u}\rangle \prec \langle\overline{a}\rangle \\ \textit{then}\ \ Q[\overline{u};\ \overline{f}(\overline{u})] \end{bmatrix}$		

which is an abbreviation of

$(\forall\ \overline{u})^{\exists}\begin{bmatrix} \textit{if}\ \ \overline{obj}(\overline{u}) \\ \textit{then}\ \ \textit{if}\ \ \langle\overline{u}\rangle \prec \langle\overline{a}\rangle \\ \qquad \textit{then}\ \ Q[\overline{u};\ \overline{f}(\overline{u})] \end{bmatrix}$		

The universal quantifier $(\forall\ \overline{u})$ is of strict existential force, as indicated by its annotation. Therefore, by the $\exists$-*elimination* rule, we may drop this quantifier, to obtain

$\textit{if}\ \ \overline{obj}(\overline{u})$ $\textit{then}\ \ \textit{if}\ \ \langle\overline{u}\rangle \prec \langle\overline{a}\rangle$ $\qquad \textit{then}\ \ Q[\overline{u};\ \overline{f}(\overline{u})]$		

In fact, whenever we want to construct a recursive program, we shall automatically remove the quantifier and include the above row in our initial tableau. ∎

THE INDUCTION RULE

The well-founded induction principle is used in forming the initial tableau for a recursive program, in which we include the induction hypothesis among our initial assertions. Nevertheless, we may wish to use the principle at subsequent stages of the derivation. For this purpose, we can apply the usual induction rules, which, extended, have no effect on the output entries. We present only the extended well-founded induction rule.

Rule (well-founded induction)

assertions	goals	$\overline{f}(\overline{a})$
	$(\forall\ obj\ x)\mathcal{F}[x]$	$\overline{s}$
$obj(r)$		$\overline{s}$
if $obj(u)$ *then if* $u \prec r$ *then* $\mathcal{F}[u]$		$\overline{s}$
	$\mathcal{F}(r)$	$\overline{s}$

Here the required goal is a closed sentence, obj is a unary predicate symbol, $\prec$ is a well-founded relation over obj, and r is a new constant. ∎

14.7 REVIEW OF PROGRAM SYNTHESIS

At this point we review our basic synthesis before presenting examples of the derivations of specific programs.

In a chosen theory, we are given a specification $Q[\overline{x}; \overline{z}]$, with input sorts $\overline{obj}$, where $\overline{x}$ and $\overline{z}$ are the input and output variables, respectively. We would like to

construct a program $\overline{f}(\overline{x}) = \overline{t}[\overline{x}]$ that satisfies this specification, in the sense that the *correctness* condition

$$\begin{aligned} &\textit{if } (\forall\, \overline{obj}\ \overline{x})\,[\,\overline{f}(\overline{x}) = \overline{t}[\overline{x}]\,] \\ &\textit{then } (\forall\, \overline{obj}\ \overline{x})\,Q[\overline{x};\ \overline{f}(\overline{x})] \end{aligned}$$

is valid in the theory.

If we want to exclude recursive programs, we take the following initial tableau:

assertions	goals	$\overline{f}(\overline{a})$
$\overline{obj}(\overline{a})$		
	$Q[\overline{a};\ \overline{z}]$	$\overline{z}$

We include n output columns, one for each output variable $\overline{z} = z_1, z_2, \ldots, z_n$.

If we want to allow recursive programs, we take the following initial tableau:

assertions	goals	$\overline{f}(\overline{a})$
$\overline{obj}(\overline{a})$		
if $obj\,(\overline{u})$ *then if* $(\overline{u}) \prec (\overline{a})$ *then* $Q\,[\overline{u};\ \overline{f}(\overline{u})]$		
	$Q[\overline{a};\ \overline{z}]$	$\overline{z}$

Whether we allow recursive programs or not, we may include any valid sentences of the theory as assertions of the tableau.

We include in the primitive list the function symbols $\overline{f}$ and any other symbols that are permitted to occur in the final program.

To derive a program, we successively apply extended deduction rules to the initial tableau. These rules add new rows to the tableau while maintaining validity and satisfying terms. The derivation must continue until we obtain the final assertion *false* or the final goal *true*, whose output entries $\overline{t}'[\overline{a}]$ are all primitive expressions. At this point, we may stop the derivation.

Let $\overline{t}[\overline{a}]$ be obtained from $\overline{t}'[\overline{a}]$ by replacing any variable with an arbitrary primitive constant. Then the final program we obtain is

$$\overline{f}(\overline{x}) \;=\; \overline{t}[\overline{x}].$$

We have shown that this program will satisfy the specification.

Even if we obtain a final row with primitive output entries, we may choose to continue the derivation. If we again derive the final assertion *false* or the final goal *true*, it may have different primitive output entries. In this case, we obtain a different program meeting the same specification.

Once we have derived a program, we may use it in deriving other programs. We extend our theory by introducing the new axiom

$$\begin{array}{l} \textit{if } \overline{obj}(\overline{x}) \\ \textit{then } \overline{f}(\overline{x}) = \overline{t}[\overline{x}] \end{array}$$

This axiom may be included as an assertion in the initial tableau for future derivations. We may also include the assertion that the program does meet its specification,

$$\begin{array}{l} \textit{if } \overline{obj}(\overline{x}) \\ \textit{then } Q[\overline{x}; \overline{f}(\overline{x})] \end{array}$$

By our *correctness* condition, this is a valid sentence of the extended theory.

Now that we have reviewed the derivation process, let us illustrate it with the derivation of a program.

FULL EXAMPLE: REDHEAD

In this section, we present the full derivation of the *redhead* program; fragments of this derivation have already been presented. In a *family* theory, for given persons x_1, x_2, and x_3, we are told that x_1 is a parent of x_2, that x_2 is a parent of x_3, and that x_1 is redheaded but x_3 is not (we are not told the hair color of x_2). We are asked to construct two programs, $rh(x_1, x_2, x_3)$ and $nrh(x_1, x_2, x_3)$, to yield two persons z_1 and z_2, respectively, such that z_1 is a parent of z_2 and that z_1 is redheaded but z_2 is not. In short, we are given the specification

$$\begin{array}{l|l} & \textit{if } par(x_1, x_2) \textit{ and } par(x_2, x_3) \textit{ and} \\ & \quad red(x_1) \textit{ and not } red(x_3) \\ Q[x_1, x_2; z_1, z_2]: & \textit{then } par(z_1, z_2) \textit{ and} \\ & \quad red(z_1) \textit{ and not } red(z_2) \end{array}$$

The input sort for each of the three inputs is $\overline{person}$. Here *par* and *red* are primitive predicate symbols. In this theory, every element is of sort *person*. This is expressed by the simplification

$$person(u) \Rightarrow true.$$

(If we were dealing with a combined theory, we would not include this simplification.)

We shall derive a nonrecursive program. ⟪ capital heads, A, G numbers ⟫

The initial tableau is:

assertions	goals	$rh(a_1, a_2, a_3)$	$nrh(a_1, a_2, a_3)$
$A1.$ $person(a_1)$ *and* $person(a_2)$ *and* $person(a_3)$			
	$G2.$ *if* $par(a_1, a_2)$ *and* $par(a_2, a_3)$ *and* $red(a_1)$ *and not* $red(a_3)$ *then* $par(z_1, z_2)$ *and* $red(z_1)$ *and not* $red(z_2)$	z_1	z_2

Note that the initial assertion $A1$ is immediately simplified to the trivial assertion *true*. By the *if-split* rule, followed by the *and-split* rule:

$A3.$ $\boxed{par(a_1, a_2)}^-$			
$A4.$ $par(a_2, a_3)$			
$A5.$ $red(a_1)$			
$A6.$ *not* $red(a_3)$			
	$G7.$ $\boxed{par(z_1, z_2)}^+$ *and* $red(z_1)$ *and not* $red(z_2)$	z_1	z_2

Note that the output entries z_1 and z_2 have been dropped from the assertions, because they have no occurrences of these variables.

By the resolution rule applied to $A3$ and $G7$, $\{z_1 \leftarrow a_1, z_2 \leftarrow a_2\}$:

	$G8.$ $\boxed{red(a_1)}^+$ *and not* $red(a_2)$	a_1	a_2

Note that no conditional terms were formed in the output entries, because only one of the required goals has output entries.

Recall

*A*5. $\boxed{red(a_1)}^-$			

By the resolution rule { },

	*G*9. *not red* (a_2)	a_1	a_2

According to this row, in the case in which a_2 is red-headed, a_1 and a_2 will be suitable outputs for $rh\,(a_1, a_2, a_3)$ and $nrh\,(a_1, a_2, a_3)$, respectively. Let us set this row aside for a while.

Recall assertion *A*4 and goal *G*7:

*A*4. $\boxed{par\,(a_2, a_3)}^-$			
	*G*7. $\boxed{par\,(z_1, z_2)}^+$ *and* $red\,(z_1)$ *and not* $red\,(z_2)$	z_1	z_2

By the resolution rule, $\{z_1 \leftarrow a_2,\ z_2 \leftarrow a_3\}$:

	*G*10. $red(a_2)$ *and* $\boxed{not\ red(a_3)}^+$	a_2	a_3

Recall assertion *A*6:

*A*6. $\boxed{not\ red(a_3)}^-$			

By the resolution rule, { }

	*G*11. $\boxed{red\,(a_2)}^+$	a_2	a_3

Recall

	*G*9. *not* $\boxed{red\,(a_2)}^-$	a_1	a_2

By the resolution rule, { }:

	$G11.$ *true*	*if* $red(a_2)$ *then* a_2 *else* a_1	*if* $red(a_2)$ *then* a_3 *else* a_2

Note the conditional terms have been formed in the output entries, because both required goals have distinct output entries.

We have derived the final goal *true* with primitive output entries. Therefore we may stop the derivation and obtain a final program.

$$rh(x_1, x_2, x_3) = \begin{cases} \textit{if } red(x_2) \\ \textit{then } x_2 \\ \textit{else } x_1 \end{cases}$$

and

$$nrh(x_1, x_2, x_3) = \begin{cases} \textit{if } red(x_2) \\ \textit{then } x_3 \\ \textit{else } x_2 \end{cases}$$

14.8 FULL EXAMPLE: FRONT-LAST

⟪ should this be a section or a subsection? ⟫

In this section, we present the full derivation of a *front-last* program to find the string of all but the last character in a nonempty string, and the last character itself (see Problem [I]7.5).

In the theory of strings, we are given the specification

$$Q[x;\, z_1,\, z_2]: \quad \begin{array}{l} \textit{if } \textit{not } (x = \Lambda) \\ \textit{then } \textit{string}(z_1) \textit{ and } \textit{char}(z_2) \textit{ and} \\ \qquad x = z_1 * z_2 \end{array}$$

with input sort *string*. ⟪ box? ⟫

We shall derive a recursive program. Because there is only one input, ⟪ MB: out?? ⟫ we may use a well-founded relation over *obj*, that is, over *string*, rather than over tuples of strings; in this case, we take $\prec$ to be $\prec_{tail}$, the *tail* relation over strings. This is defined by the axiom

$$(\forall\ \textit{string}\ u, v) \left[\begin{array}{l} u \prec_{tail} v \\ \quad \equiv \\ (\exists\ \textit{char}\ w)[w * u = v] \end{array} \right]$$

⟪ have we said this is well-founded? ⟫

Because we are working in a pure theory of strings, rather than a combined theory, we know that every element is a string. This is expressed in the simplification

$$string(u) \Rightarrow true.$$

Our initial tableau is therefore

assertions	goals	$front(a)$	$last(a)$
$A1.$ *if* $u \prec_{tail} a$ *then if not* $(u = \Lambda)$ *then* $char(last(u))$ *and* $u = front(u) * last(u)$			
	$G2.$ *if not* $(a = \Lambda)$ *then* $char(z_2)$ *and* $a = z_1 * z_2$	z_1	z_2

Note that the rows have been transformed by the simplification $string(u) \Rightarrow u$.

We include in the primitive list the basic symbols of the theory of strings (Λ, *head*, *tail*, *char*) as well as the function symbols *front* and *last* themselves.

We include as assertions in the initial tableau certain axioms and valid sentences of the theory of strings, including the above definition of the *tail* relation.

We shall use the property

$$(\forall\ string\ u, v) \begin{bmatrix} if\ \ not\ (u = \Lambda) \\ then\ \ tail(u) \prec_{tail} u \end{bmatrix} \qquad (tail)$$

⟪ out? ⟫

We shall need an assertion expressing the *trichotomy* property

$$(\forall\ string\ u)[u = \Lambda\ \ or\ \ char(u)\ \ or\ \ not\ (tail(u) = \Lambda)] \quad (trichotomy)$$

that is, every string is either empty, consists of a single character, or has a nonempty tail.

We can immediately decompose the goal.

By the *if-split* rule applied to $G2$:

A3. *not* $(a = \Lambda)$			
	G4. *char*(z_2) *and* $a = z_1 * z_2$	z_1	z_2

Note that the output entries z_1 and z_2 have been dropped from row *A3*, because they do not occur free in the assertion.

THE CHARACTER CASE

We begin by deriving the portion of the program corresponding to the case in which a is a single character. We focus on our goal *G4* and use an axiom for concatenation.

Recall the ***left-empty*** axiom for concatenation:

$\left[\boxed{\Lambda * u} = u\right]^-$			

Recall goal *G4*.

	G4. *char*(z_2) *and* $a = \boxed{z_1 * z_2}$	z_1	z_2

By the ***equality*** rule, $\{z_1 \leftarrow \Lambda, u \leftarrow z_2\}$:

	G5. *char*(z_2) *and* $\boxed{a = z_2}^+$	Λ	z_2

Recall the ***reflexivity*** axiom for equality:

$\boxed{u = u}^-$			

By the ***resolution*** rule, $\{u \leftarrow a, z_2 \leftarrow a\}$:

	G6. *char*(a)	Λ	a

Note that, by this stage, the output entries for $front(a)$ and $last(a)$ have been chosen to be the terms Λ and a, respectively. In other words, when $char(a)$ is true, Λ and a will be suitable outputs.

INTRODUCTION OF THE RECURSIVE CALL

Let us set aside goal $G6$ for a while and return our attention to goal $G4$. We again use an axiom for concatenation.

Recall the *left-prefix* axiom for concatenation:

if $char(w)$ *then* $\left[\boxed{(w \cdot u) * v} = w \cdot (u * v)\right]^-$			

Recall goal $G7$:

	$G4.$ $char(z_2)$ *and* $a = \boxed{z_1 * z_2}$	z_1	z_2

By the *equality* rule, $\{z_1 \leftarrow w \cdot u, v \leftarrow z_2\}$, rename u to z_1':

	$G7.$ $char(w)$ *and* $char(z_2)$ *and* $a = w \cdot \boxed{z_1' * z_2}$	$w \cdot z_1'$	z_2

As a result of this step, the output entry for $front(a)$ has been taken to be $w \cdot z_1'$, where w and z_1' have yet to be selected. We now use our induction hypothesis twice in succession.

Recall our induction hypothesis:

$A1.$ *if* $u \prec_{tail} a$ *then if not* $(u = \Lambda)$ *then* $char(last(u))$ *and* $\left[u = \boxed{front(u) * last(u)}\right]^-$			

By the *equality* rule, right-to-left, $\{z_1' \leftarrow front(u), z_2 \leftarrow last(u)\}$:

	G8. $not \begin{bmatrix} if\ u \prec_{tail} a \\ then\ u = \Lambda \end{bmatrix}$ *and* $char(w)$ *and* $\boxed{char(last(u))}^{+}$ *and* $a = w \circ u$	$w \circ front(u)$	$last(u)$

Recall our induction hypothesis again (renaming u to u'):

A1. *if* $u' \prec_{tail} a$ *then if not* $(u' = \Lambda)$ *then* $\boxed{char(last(u'))}^{-}$ *and* $u' = front(u') \circ last(u')$			

By the *resolution* rule, $\{u \leftarrow u'\}$:

	G9. $not \begin{bmatrix} if\ u' \prec_{tail} a \\ then\ u' = \Lambda \end{bmatrix}$ *and* $char(w)$ *and* $\boxed{a = w \circ u'}^{+}$	$w \circ front(u')$	$last(u')$

As a result of this step, the output entries for $front(a)$ and $last(a)$ have been taken to be $w \circ front(u')$ and $last(u')$, respectively. In other words, we have introduced recursive calls into the program. Note that the subsentence

$$not\ (if\ u' \prec_{tail} a\ then\ u' = \Lambda)$$

is propositionally equivalent to

$$u' \prec_{tail} a\ and\ not\ (u' = \Lambda).$$

Let us rewrite it that way for clarity, although which form we use has no bearing on the rest of the derivation.

At the next stage, the input a is decomposed.

Recall the *decomposition* property of *head* and *tail*:

if not $(u = \Lambda)$ *then* $\boxed{u = head(u) \circ tail(u)}^{-}$			

By the *resolution* rule, $\{u \leftarrow a, w \leftarrow head(a), u' \leftarrow tail(a)\}$:

	G10. *not* $(a = \Lambda)$ *and* $tail(a) \prec_{tail} a$ *and* *not* $(tail(a) = \Lambda)$ *and* $\boxed{char(head(a))}^{+}$	$head(a) \bullet front(tail(a))$	$last(tail(a))$

Recall the *sort* property of *head*:

if *not* $(u = \Lambda)$ *then* $\boxed{char(head(u))}^{-}$			

By the *resolution* rule, $\{u \leftarrow a\}$:

	G4. *not* $(a = \Lambda)$ *and* $tail(a) \prec_{tail} a$ *and* *not* $(tail(a) = \Lambda)$	$head(a) \bullet front(tail(a))$	$last(tail(a))$

FINISHING THE NON-CHARACTER CASE

We now show that the well-founded relation $\prec_{tail}$ does indeed hold between the argument $tail(a)$ and the original input a. We then use goal G9, which we had developed and set aside.

Recall the *tail* property of the *tail* relation:

if *not* $(u = \Lambda)$ *then* $\boxed{tail(u) \prec_{tail} u}^{-}$		

Recall goal G11:

	G11. *not* $(a = \Lambda)$ *and* $\boxed{tail(a) \prec_{tail} a}^{+}$ *and* *not* $(tail(a) = \Lambda)$	$head(a) \bullet front(tail(a))$	$last(tail(a))$

⟪ repeat $G11$? ⟫

By the *resolution* rule, $\{u \leftarrow a\}$:

	$G12.$ $not\ (a = \Lambda)\ and$ $\boxed{not\ (tail(a) = \Lambda)}^{+}$	$head(a) \bullet$ $front(tail(a))$	$last(tail(a))$

Recall the *trichotomy* property:

$u = \Lambda\ or\ char(u)\ or\ \boxed{not(tail(u) = \Lambda)}^{-}$			

By the *resolution* rule, $\{u \leftarrow a\}$:

	$G13.$ $not\left(\boxed{a = \Lambda}^{-}\ or\ char(a)\right)$ and $not\ \boxed{a = \Lambda}^{-}$	$head(a) \bullet$ $front(tail(a))$	$last(tail(a))$

Recall assertion $A3$:

$A6.\ not\ \boxed{a = \Lambda}^{+}$			

By the *resolution* rule { }:

	$G14.\ not\ char(a)$	$head(a) \bullet$ $front(tail(a))$	$last(tail(a))$

COMBINING THE TWO CASES

Recall our previous goals $G6$ and $G14$:

	$G6.\ \boxed{char(a)}^{+}$	Λ	a
	$G14.\ not\ \boxed{char(a)}^{-}$	$head(a) \bullet$ $front(tail(a))$	$last(tail(a))$

By the *resolution* rule, { }:

	$G18.\ \ true$	*if* $char(a)$ *then* Λ *else* $head(a) \bullet$ $front(tail(a))$	*if* $char(a)$ *then* a *else* $last(tail(a))$

At this point conditional terms have been formed in the output column, because the goals $G9$ and $G17$ both have output entries.

We have derived the final goal *true* with primitive output entries. Therefore, we may stop the derivation and obtain the final program

$$front(x) = \begin{cases} if\ \ char(x) \\ then\ \ \Lambda \\ else\ \ head(x) \bullet front(tail(x)) \end{cases}$$

and

$$last(x) = \begin{cases} if\ \ char(x) \\ then\ \ x \\ else\ \ last(tail(x)) \end{cases}$$

⟪ exercise: reversing a string. See SOL14 file ⟫

14.9 FULL EXAMPLE: FLATTENING A TREE

In this section, we present the full derivation of a program for flattening a tree. This is actually an example of program transformation, because we are given one method of computing a function and we derive another one. The reader will recall that we have defined by the following axioms a function *flattree* (Section [I]8.4), which takes a tree as its argument and yields the string of all its atoms:

$$(\forall\ atom\ u)\big[flattree(u) = u\big] \qquad (atom)$$

$$(\forall\ tree\ u, v)\big[flattree(u \circ v) = flattree(u) * flattree(v)\big]\ (construction)$$

Here $\circ$ is the tree construction function and $*$ is the string concatenation function.

These axioms are computationally suggestive; they provide a method for computing the function.

This example illustrates two points. First, we have observed that, in proving a theorem, it may be necessary to prove a more general theorem, so as to have the benefit of a stronger induction hypothesis. This often occurs when the proof is part of the derivation of a program. In that case, the program we derive from the proof of the more general theorem is used as a "subprogram" by the main program, which we derive from the proof of the given theorem.

The example also illustrates the use of a combined theory in program derivation. The program is applied to a tree and yields a string; therefore, we must work in a combined theory of trees and strings. In the combined theory, we identify the atoms of the trees with the characters of the strings; this is expressed by the axiom

$$(\forall\, u)[\mathit{char}(u) \equiv \mathit{atom}(u)] \qquad (\textit{character-atom})$$

The specification for the new program *flattree*1(x) is simply

$$Q_1[x; z]: \quad z = \mathit{flattree}(x)$$

In other words, the *flattree*1 program is to yield the same result as the given *flattree* program. We shall take *obj* to be *tree*.

We shall not include the function symbol *flattree* itself in the primitive list. This will ensure that we cannot obtain a *flattree*1 program that relies on the *flattree* program. We shall also omit the concatenation function symbol $*$ from the primitive list; this will force us to express the new program in terms of the prefix function $\cdot$ rather than the less efficient concatenation function.

To conduct this derivation, we must first derive a more general program *flattree*2(x_1, x_2), to meet the specification

$$Q_2[x_1, x_2; z]: \quad z = \mathit{flattree}(x_1) * x_2$$

We take x_1 to be a tree and x_2 to be a string; that is, our input sorts are *tree* and *string*, respectively. ⟪ out? ⟫ This generalization step is not done by any rule of the system; we assume that the generalized specification is supplied to us.

Before we begin the derivation of *flattree*2, let us see how it will enable us to complete the derivation of *flattree*1.

THE DERIVATION OF FLATTREE1

The program for *flattree*1 will not be recursive. We may therefore take our initial tableau without an induction hypothesis, as follows:

assertions	goals	*flattree*1(a)
$A2.$ *tree*(a)		
	$G2.$ z = *flattree* (a)	z

Because our theory is a combined theory of trees and strings, rather than a pure theory, we must retain sort conditions, such as *tree* (a), to distinguish between the two sorts of elements *tree* and *string*.

Assuming that the derivation of *flattree*2 has been successful, we may include in the initial tableau for *flattree*1 an assertion that *flattree*2 does indeed meet its specification, namely

$A3.$ *if* *tree* (x_1) *and* *string*(x_2) *then* *flattree*2(x_1, x_2) = *flattree* (x_1) $*$ x_2		

For this derivation, we include *flattree*2 but not *flattree*1 or *flattree* in the primitive list.

We first obtain a special case of the above assertion by invoking a property of concatenation. This result will be useful in establishing the initial goal.

Recall the *right-empty* property of concatenation:

if *string*(u) *then* $\left[\boxed{u * \Lambda} = u\right]^{-}$		

Recall assertion $A3$:

$A3.$ *if* *tree*(x_1) *and* *string*(x_2) *then* *flattree*2(x_1, x_2) = $\boxed{\textit{flattree}\,(x_1) * x_2}$		

By the *equality* rule, $\{u \leftarrow$ *flattree* $(x_1), x_2 \leftarrow \Lambda\}$, and removal of a sort condition.

A4. $not\ (string(flattree(x_1)))$ or $[if\ tree(x_1)\ and\ string(\Lambda)$ $then\ \boxed{flattree2(x_1, \Lambda) = flattree(x_1)}^{-}]$		

Recall goal $G2$:

	$G2.\ \boxed{z = flattree(a)}^{+}$	z

By the *resolution* rule, $\{x_1 \leftarrow a, z \leftarrow flattree2(a, \Lambda)\}$, and removal of sort conditions:

	$G5.\ string(flattree(a))\ and$ $tree(a)\ and\ string(\Lambda)$	$flattree2(a, \Lambda)$

We have obtained the final goal *true* with a primitive output entry. This completes the derivation of *flattree*1; we may extract the program

$$flattree1(x) = flattree2(x, \Lambda)$$

Remark (*flattree* is not primitive)

If the function symbol *flattree* had been taken to be primitive, we could have completed the derivation more easily, but the result would not have been useful.

Recall goal $G2$ and the *reflexivity* axiom for equality:

	$G2.\ \boxed{z = flattree(a)}^{+}$	z
$\boxed{u = u}^{-}$		

By the *resolution* rule, $\{z \leftarrow flattree(a), u \leftarrow flattree(a)\}$:

	$G3'.\ true$	$flattree(a)$

If the symbol *flattree* is primitive, we may conclude the derivation, obtaining the final program

$$flattree1(x) = flattree(x).$$

This program satisfies the specification $z = \mathit{flattree}(x)$, of course, but we have not transformed the given program *flattree* as we had intended. When *flattree* is excluded from the primitive list, we are prevented from stopping the derivation at this stage. ∎

THE DERIVATION OF FLATTREE2

Now let us present the derivation of *flattree*2. We are given the specification

$$Q_2[x_1, x_2; z]: \quad z = \mathit{flattree}(x_1) * x_2$$

where the input sorts are *tree* and *string*, respectively.

The program for *flattree*2 will be recursive. Because there are two inputs, a tree and a string, we must take our well-founded relation $\prec$ to be over 2-tuples (that is, pairs) of sort *tree* and *string*. In this case, we shall take $\prec$ to be $\prec_{\pi_1(\mathit{child})}$, the first projection of the *child* relation, defined by the axiom

$$\begin{array}{|l} (\forall\ \mathit{tree}\ u_1,\ v_1)(\forall\ \mathit{string}\ u_2,\ v_2)\left[\begin{array}{l}\langle u_1,\ u_2\rangle \prec_{\pi_1(\mathit{child})} \langle v_1, v_2\rangle \\ \quad \equiv \\ u_1 \prec_{\mathit{child}} v_1\end{array}\right] \qquad (\textit{first projection}) \\ \end{array}$$

where the child relation $\prec_{\mathit{child}}$ is defined by the axiom

$$\begin{array}{|l} (\forall\ \mathit{tree}\ u,\ v)\left[\begin{array}{l}u \prec_{\mathit{child}} v \\ \quad \equiv \\ \left[\begin{array}{l}(\exists\ \mathit{tree}\ w)[u * w = v] \\ \quad or \\ (\exists\ \mathit{tree}\ w)[w * u = v]\end{array}\right]\end{array}\right] \qquad (\textit{child}) \\ \end{array}$$

In other words, the two subtrees *left* (v) and *right* (v) are the two children of the nonatomic tree v. ⟪ shown to be well-founded? ⟫

Our initial tableau is:

assertions	goals	*flattree2(*
*A*1. *tree*(a_1) *and* *string*(a_2)		
*A*2. *if* $\langle u_1, u_2\rangle \prec_{\pi_1(child)} \langle a_1, a_2\rangle$ *then if* *tree*(u_1) *and* *string*(u_2) *then* *flattree2*(u_1, u_2) = *flattree*$(u_1) * u_2$		
	*G*3. z = *flattree*$(a_1) * a_2$	z

Recall that we include *flattree2* itself in the primitive list, but neither *flattree* nor the concatenation function x^*y.

THE ATOMIC CASE

We begin with the portion of the derivation that pertains to the case in which the first input a_1 is an atom. We focus our attention on the initial goal.

Recall the *atom* axiom for *flattree*:

if *atom*(u) *then* $\left[\boxed{\textit{flattree}(u)} = u\right]^-$		
	*G*3. z = $\boxed{\textit{flattree}(a_1)} * a_2$	z

By the *equality* rule, $\{u \leftarrow a_1\}$:

	*G*4. *atom*(a_1) *and* z = $\boxed{a_1^* a_2}$	z

Recall the *character* property of $*$:

if *char*(u) *and* *string*(v) *then* $\left[\boxed{u * v} = u \bullet v\right]^-$		

By the *equality* rule, $\{u \leftarrow a_1, v \leftarrow a_2\}$ and removal of sort conditions:

	$G5.$ $char(a_1)$ *and* $atom(a_1)$ *and* $\boxed{z = a_1 \bullet a_2}^{+}$	z

Recall the *reflexivity* axiom for equality:

$\boxed{u = u}^{-}$		

By the *resolution* rule, $\{u \leftarrow a_1 \bullet a_2,\ z \leftarrow a_1 \bullet a_2\}$:

	$G6.$ $\boxed{char(a_1)}$ *and* $atom(a_1)$	$a_1 \bullet a_2$

As a result of this step, the output has been taken to be $a_1 \bullet a_2$.

Recall the *character-atom* axiom for the combined theory:

$\left[\boxed{char(u)} \equiv atom(u)\right]^{-}$		

By the *equivalence* rule, $\{u \leftarrow a_1\}$:

	$G7.$ $atom(a_1)$	$a_1 \bullet a_2$

Let us set this goal aside for a while.

DECOMPOSITION OF THE INPUT

The balance of the derivation concerns the case in which a_1 is nonatomic. In this case, we may decompose a_1 into its two components.

Recall goal $G3$:

	$G3.$ $z = flattree\left(\boxed{a_1}\right) * a_2$	z

Recall the *decomposition* property of trees:

if tree(u) *then if not atom*(u) *then* $\left[\boxed{u} = \mathit{left}(u) \circ \mathit{right}(u)\right]^{-}$		

By the ***equality*** rule, $\{u \leftarrow a_1\}$, and removal of a sort condition:

	G8. ***not atom***(a_1) ***and*** $z = \boxed{\mathit{flattree}(\mathit{left}(a_1) \circ \mathit{right}(a_1))} * a_2$	z

Recall the ***constructor*** axiom for ***flattree***:

if tree(u) ***and*** *tree*(v) *then* $\left[\boxed{\mathit{flattree}(u \circ v)} = \mathit{flattree}(u) * \mathit{flattree}(v)\right]^{-}$		

By the ***equality*** rule, $\{u \leftarrow \mathit{left}(a_1), v \leftarrow \mathit{right}(a_1)\}$, and removal of a sort condition:

	G9. ***not atom***(a_1) ***and*** $z = \boxed{(\mathit{flattree}(\mathit{left}(a_1)) * \mathit{flattree}(\mathit{right}(a_1))) * a_2}$	z

Recall the ***associativity*** property of concatenation:

if string(u) ***and*** *string*(v) ***and*** *string*(w) *then* $\left[\boxed{(u * v) * w} = u * (v * w)\right]^{-}$		

By the ***equality*** rule, $\{u \leftarrow \mathit{flattree}(\mathit{left}(a_1)), v \leftarrow \mathit{flattree}(\mathit{right}(a_1)), w \leftarrow a_2\}$, and removal of sort conditions:

	G10. ***not atom***(a_1) ***and*** $z = \mathit{flattree}(\mathit{left}(a_1)) * (\mathit{flattree}(\mathit{right}(a_1)) * a_2)$	z

INTRODUCTION OF THE RECURSIVE CALL

We are now in a position to use our induction hypothesis. This will result in the

appearance of a recursive call in the output column.

Recall the induction hypothesis:

$A2.$ *if* $\langle u_1, u_2\rangle \prec_{\pi_1(child)} \langle a_1, a_2\rangle$ *then if* $tree(u_1)$ *and* $string(u_2)$ *then* $\boxed{flattree2(u_1, u_2) = \boxed{flattree(u_1) * u_2}}^-$		

Recall goal G10:

	$G10.$ *not* $atom(a_1)$ *and* $z = flattree(left(a_1)) * \boxed{flattree(right(a_1)) * a_2}$	z

By the *equality* rule, right-to-left, $\{u_1 \leftarrow right(a_1), u_2 \leftarrow a_2\}$, and removal of sort conditions:

	$G11.$ $\langle right(a_1), a_2\rangle \prec_{\pi_1(child)} \langle a_1, a_2\rangle$ *and* *not* $atom(a_1)$ *and* $\boxed{z = flattree(left(a_1)) * flattree2(right(a_1), a_2)}^+$	z

Recall the induction hypothesis, again:

$A2.$ *if* $\langle u_1, u_2\rangle \prec_{\pi_1(child)} \langle a_1, a_2\rangle$ *then if* $tree(u_1)$ *and* $string(u_2)$ *then* $\boxed{flattree2(u_1, u_2) = flattree(u_1) * u_2}^-$		

By the *resolution* rule, $\{u_1 \leftarrow left(a_1),\ u_2 \leftarrow flattree2(right(a_1), a_2),\ z \leftarrow flattree2(left(a_1), flattree2(right(a_1), a_2))\}$, and removal of sort conditions:

	$G12.$ $\left[\begin{array}{l}\langle left(a_1),\ flattree2(right(a_1),\ a_2)\rangle \\ \quad \prec_{\pi_1(child)} \\ \langle a_1, a_2\rangle\end{array}\right]$ and $string(flattree2(right(a_1),\ a_2))$ and $\langle right(a_1),\ a_2\rangle \prec_{\pi_1(child)} \langle a_1,\ a_2\rangle$ and $not\ atom(a_1)$	$flattree2(left(a_1),$ $flattree2(right(a_1),\ a_2))$

Note that, as a result of this step, we have introduced two nested recursive calls into the output entry.

ESTABLISHING THE WELL-FOUNDED RELATION

We now show that the argument pairs for the recursive calls are "less than" the given argument pair $\langle a_1, a_2\rangle$, with respect to our selected well-founded relation $\prec_{\pi_1(child)}$. We use properties of the first-projection and *child* relations.

Recall the definition of the first-projection relation and goal $G12$:

$if\ tree(u_1)\ and\ string(u_2)\ and$ $tree(v_1)\ and\ string(v_2)$ $then\ \left[\begin{array}{l}\langle u_1, u_2\rangle \prec_{\pi_1(child)} \langle v_1, v_2\rangle \\ \equiv \\ u_1 \prec_{child} v_1\end{array}\right]$		
	$G12.$ $\left[\begin{array}{l}\langle left(a_1),\ flattree2(right(a_1),\ a_2)\rangle \\ \quad \prec_{\pi_1(child)} \\ \langle a_1, a_2\rangle\end{array}\right]$ and $string(flattree2(right(a_1),\ a_2))$ and $\langle right(a_1),\ a_2\rangle \prec_{\pi_1(child)} \langle a_1,\ a_2\rangle$ and $not\ atom(a_1)$	$flattree2(left(a_1),$ $flattree2(right(a_1),\ a_2))$

By the *equivalence* rule, $\{u_1 \leftarrow left(a_1),\ u_2 \leftarrow flattree2(right(a_1),\ a_2),\ v_1 \leftarrow a_1,\ v_2 \leftarrow a_2\}$, and removal of sort conditions:

	G13. $left(a_1) \prec_{child} a_1$ and $string(flattree2(right(a_1), a_2))^+$ $\langle right(a_1), a_2\rangle \prec_{\pi_1(child)} \langle a_1, a_2\rangle$ and not $atom(a_1)$	$flattree2(left(a_1),$ $flattree2(right(a_1), a_2))$

Contrary to our habit, we have not automatically removed the *sort* condition $string(flattree2(right(a_1), a_2))$; the removal of this condition is unusual in that it requires the induction hypothesis.

Recall the induction hypothesis:

A2.	if $\langle u_1, u_2\rangle \prec_{\pi_1(child)} \langle a_1, a_2\rangle$ then if $tree(u_1)$ and $string(u_2)$ then $string(flattree2(u_1, u_2))^-$ and $flattree2(u_1, u_2) = flattree(u_1) * u_2$		

By the resolution rule, $\{u_1 \leftarrow right(a_1), u_2 \leftarrow a_2\}$, and removal of *sort* conditions:

	G14. $left(a_1) \prec_{child} a_1$ and $\langle right(a_1), a_2\rangle \prec_{\pi_1(child)} \langle a_1, a_2\rangle$ and not $atom(a_1)$	$flattree2(left(a_1),$ $flattree2(right(a_1), a_2))$

We now invoke properties of the first-projection and *child* relations.

Recall the definition of the first-projection relation:

if $tree(u_1)$ and $string(u_2)$ and $tree(v_1)$ and $string(v_2)$ then $[\langle u_1, u_2\rangle \prec_{\pi_1(child)} \langle v_1, v_2\rangle \equiv u_1 \prec_{child} v_1]^-$			

By the *equivalence* rule, $\{u_1 \leftarrow right(a_1), u_2 \leftarrow a_2, v_1 \leftarrow a_1, v_2 \leftarrow a_2\}$, and removal of sort conditions:

	G15. $\boxed{left(a_1) \prec_{child} a_1}^{+}$ *and* $\boxed{right(a_1) \prec_{child} a_1}^{+}$ *and* *not* $atom(a_1)$	$flattree2(left(a_1), flattree2(right(a_1), a_2))$

Recall the *left* and *right* properties of the *child* relation:

if $tree(u)$ *and not* $atom(u)$ *then* $\boxed{left(u) \prec_{child} u}^{-}$		
if $tree(u)$ *and not* $atom(u)$ *then* $\boxed{right(u) \prec_{child} u}^{-}$		

By two applications of the *resolution* rule, $\{u \leftarrow a_1\}$, and removal of sort conditions:

	G16. *not* $atom(a_1)$	$flattree2(left(a_1), flattree2(right(a_1), a_2))$

THE FINAL STAGE

We now use the earlier goal, which we developed and set aside.

Recall goals *G*16 and *G*7:

	G16. *not* $\boxed{atom(a_1)}^{-}$	$flattree2(left(a_1), flattree2(right(a_1), a_2))$
	G7. $\boxed{atom(a_1)}^{+}$	$a_1 \cdot a_2$

By the *resolution* rule, { }:

	$G17.$ *true*	*if* $atom(a_1)$ *then* $a_1 \circ a_2$ *else* $flattree2(left(a_1),$ $\quad flattree2(right(a_1), a_2))$

At this stage a conditional term has been introduced into the output column.

We have obtained the final goal *true* with a primitive output entry. Therefore, we may conclude the derivation and obtain the program

$$flattree2(x_1, x_2) \;=\; \begin{cases} \textit{if } atom(x_1) \\ \textit{then } x_1 \circ x_2 \\ \textit{else } flattree2(left(x_1),\ flattree2(right(x_1), x_2)) \end{cases}$$

This program, combined with the program we obtained from the first derivation,

$$flattree1(x_1) \;=\; flattree2(x_1, \Lambda)$$

gives us an alternative method for computing the *flattree* function.

PROBLEMS

Problem (sex)

Consider the extension of the family theory, in which every person is either male or female, that is,

$$(\forall\ person\ u)[sex(u,\ male)\ \ or\ \ sex(u,\ female)] \qquad (sex)$$

In this theory, give the full derivation of a program $s(x) = t[x]$ to find the sex of a given person, that is, to meet the specification

$$Q[x;\ z]:\quad sex(x,\ z),$$

where the input sort is *person*.

Solution (sex)

From the specification, we form the initial tableau

assertions	goals	$s(a)$
1. $person(a)$		
	2. $\boxed{sex(a, z)}^{+}$	z

the *sex* axiom:

if $person(u)$ *then* $\boxed{sex(u, male)}^{-}$ *or* $sex(u, female)$		

- By the resolution rule, $\{u \leftarrow a,\ z \leftarrow male\}$, removal of sort condition.
  ⟪ use $not(if\ \mathcal{F}\ then\ \mathcal{G}) \Rightarrow \mathcal{F}\ and\ not\ \mathcal{G}$ as a simplification? ⟫

	3. $not\ \boxed{sex(a, female)}^{-}$	$male$

- By the resolution rule, $\{z \leftarrow female\}$

	4. *true*	*if* $sex(a, female)$ *then* *female* *else* *male*

We have obtained the final goal ***true*** with a primitive output entry. Therefore we may extract the program

$$s(x) = \begin{cases} if\ sex(x, female) \\ then\ female \\ else\ male \end{cases}$$

Problem (reverse)

In the theory of strings, suppose we are given a program $reverse(u)$ to reverse a string u (Section [I]7.4) in the form of two axioms.

$$reverse(\Lambda) = \Lambda \qquad (empty)$$

$$(\forall\ string\ u, v) \begin{bmatrix} if\ char(u) \\ then\ reverse(u * v) = reverse(v) * u \end{bmatrix} \qquad (prefix)$$

Here $u * v$ is the prefix function (where u is a character) and $v_1 * v_2$ is the concatenation function.

(a) Derive a program $reverse2(x_1, y_2)$ for reversing a string x and concatenating the result and the string y.

This program must meet the specification

$$\mathcal{Q}_2[x_1, x_2; z] : \; z = reverse(x_1) * x_2$$

where the input sorts obj_1 and obj_2 are both *string*.

(b) Use this program in the derivation of a program $reverse1(x)$ (more efficient than *reverse*) for reversing a stri This program must meet the specification

$$\mathcal{Q}_1[x; z] : \; z = reverse(x)$$

The program for *reverse*2 derived in part (a) of the problem may be included in the tableau as an axiom. Also, the property that this program meets its specification, that is,

$$(\forall \; string \; u_1, \; u_2)\big[reverse2(u_1, u_2) \; = \; reverse(u_1) * u_2\big],$$

may be included.